1989

New Learning and Control Algorithms for Neural Networks.

Chung Hwa Youn
Louisiana State University and Agricultural & Mechanical College

INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book. These are also available as one exposure on a standard 35mm slide or as a 17" x 23" black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 9017311

New learning and control algorithms for neural networks

Youn, Chung Hwa, Ph.D.

The Louisiana State University and Agricultural and Mechanical Col., 1989

U·M·I

300 N. Zeeb Rd.
Ann Arbor, MI 48106

NEW LEARNING AND CONTROL ALGORITHMS

FOR NEURAL NETWORKS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

by

Chung Hwa Youn

B.S. Seoul National University, 1979
M.A. University of Texas at Austin, 1984

August 1989

Acknowledgements

I would like to thank Professor Subhash C. Kak for all of the support and assistance that he has provided over the past two years. As my advisor, he continued to help me keep focused on the objective and provided hours of discussion, review and comments.

I would also like to thank my family, especially my wife, Young, for her constant support and encouragement throughout my graduate career.

Next, I would like to thank my fellow Korean students in the Department of Computer Science and Dr. Michael C. Stinson for his emotional support and valu[illegible] discussions.

Table of Contents

	Page
1. Introduction	1
2. Background	15
2.1 Structure of a Bicameral Network	15
2.2 The Biological Basis of Neural Networks	17
2.2.1 Neurons	18
2.2.2 Synapses	20
2.2.3 Learning	20
2.2.4 Types of Memory in the Human Brain	21
2.2.5 Structure of the Human Brain	22
2.3 The Hopfield Model	22
2.4 Learning Rules	25
2.4.1 The Hebbian Rule	27
2.4.2 The Delta Rule	30
2.4.3 Convergence of the Delta Rule	33
3. A New Learning Algorithm	36
3.1 The Structure of Fixed Points	37
3.2 Symmetric Updating	38
3.3 The Correlation Continuous Unlearning Algorithm	41
3.4. Comparison of Attraction Basins	52
3.5 Comparison of Capacity	60
3.5.1 Analysis of Capacity	64
3.5.2 Two Versions of the CCU Algorithm	66
4. The Bicameral Classifiers	69

4.1 Models with Equal Probabilities for Patterns 69
4.1.1 The Hamming Network 69
4.1.2 The Image Classifier 71
4.1.3 The Bicameral Classifier 72
4.2 Models with Different Probabilities for Patterns 80
4.2.1 Pattern Classifiers 80
4.2.2 Bayes' Law 82
5. The Method of Iteration 90
5.1 The Method of Iteration with a Hidden-bit 95
5.2 The Method of Iteration with Handles 99
6. Multilayered Neural Networks 110
6.1 Bidirectional Associative Memory (BAM) 110
6.2 CCU with Bidirectional Associative Memory (CCUBAM) 113
6.3 Multilayered Feedforward Neural Networks 115
6.3.1 Backpropagation Algorithm 117
6.3.2 Layered Network as an Associative Memory 124
6.3.3 Indexing of Patterns 127
7. Conclusions 132
7.1 Neural Network System for Speech & Vision Problems 133
7.2 Directions for Future Research 137
8. References 140

List of Figures

		Page
1.	Neural Network without Hidden Neurons	9
2.	Neural Network without Hidden Neurons	9
3.	Neural Network with Hidden Neurons	10
4.	Bicameral Neural Network	16
5.	The Idealized Structure of a Neuron	19
6.	Bicameral Structure of the Human Brain	23
7.	Correlations in Asynchronous Hopfield Model	26
8.	The Hebbian Learning Rule	29
9.	The Delta Learning Rule	32
10.	The Correlation Continuous Unlearning Rule	43
11.	The Energy Structure of the Hebbian Rule	55
12.	The Energy Structure of the Delta Rule	56
13.	The Energy Structure of the CCU Rule	57
14.	The Energy Structure of Three Learning Rules	58
15.	The Method of Iteration	92
16.	Topology of the Bidirectional Associative Memory	111
17.	Performance Comparison of BAM and CCUBAM	116
18.	Activation Function of the McCulloch-Pitts Neuron	119
19.	Logistic Activation Function	119
20.	Neurons in Multilayered Feedforward Network	121
21.	Structure of the Pattern Recognition System	136

Abstract

Neural networks offer distributed processing power, error correcting capability and structural simplicity of the basic computing element. Neural networks have been found to be attractive for applications such as associative memory, robotics, image processing, speech understanding and optimization. Neural networks are self-adaptive systems that try to configure themselves to store new information. This dissertation investigates two approaches to improve performance: better learning and supervisory control. A new learning algorithm called the Correlation Continuous Unlearning (CCU) algorithm is presented. It is based on the idea of removing undesirable information that is encountered during the learning period. The control methods proposed in the dissertation improve the convergence by affecting the order of updates using a controller.

Most previous studies have focused on monolithic structures. But it is known that the human brain has a "bicameral" nature at the gross level and it also has several specialized structures. In this dissertation, we investigate the computing characteristics of neural networks that are not monolithic being enhanced by a controller that can run algorithms that take advantage of

the known global characteristics of the stored information. Such networks have been called bicameral neural networks. Stinson and Kak considered elementary bicameral models that used asynchronous control. New control methods, the method of iteration and bicameral classifier, are now proposed. The method of iteration uses the Hamming distance between the probe and the answer to control the convergence to a correct answer, whereas the bicameral classifier takes advantage of global characteristics using a clustering algorithm. The bicameral classifier is applied to two different models of equiprobable patterns as well as the more realistic situation where patterns can have different probabilities.

The CCU algorithm has also been applied to a bidirectional associative memory with greatly improved performance. For multilayered networks, indexing of patterns to enhance system performance has been studied.

Chapter One

Introduction

During the past decade, the approach of neural networks has been used by many artificial intelligence (AI) researchers who seek to achieve human-like performance in speech and image processing systems. The prime motivation for this is that neural networks offer distributed processing power, error correcting capability and structural simplicity of the basic computing element.

Traditional approaches in AI research include production systems, expert systems and frame or schema based systems, which are all rule-based structures. In a problem of increasing complexity, the only method available is to develop more rules, algorithms and more advanced searching methods. Neural networks offer a totally different approach. There are no explicitly stated rules and therefore the question of search does not arise. Furthermore, the time to solve a problem is essentially independent of problem size. Basically, neural networks are self-adaptive systems that try to configure themselves to new information (or knowledge) using "learning". When the system has to learn something, it tries to adjust the synaptic strengths of connections among neurons. It opens up the possibility that an information processing machine

can learn new information by tuning its connections. On the other hand, in a conventional AI system one must formulate explicit rules. Another difference is that in neural networks information is distributed over the system, whereas in a conventional computer, it is stored at specific addresses in its entirety.

McCulloch and Pitts [McCu43] were the first to study neuron structures for computing in 1943. The next four decades saw applications of layered neural networks for pattern recognition. An influential recent work was the 1982 paper by Hopfield [Hopf82], who proposed a model that could be worked asynchronously. In this model, a simple "sum of outer product" algorithm is used for the construction of synaptic weight matrix, which is basically the "Hebbian" rule of learning. It has two modes of operation. In the asynchronous mode, the network always converges to a fixed point, whereas in the faster synchronous operation, there is no guarantee of a fixed point. In terms of performance, about 0.15N memories, where N is the number of neurons, can be simultaneously remembered before error in retrieval is severe. Some of the inherent drawbacks of the model are listed below.

1. The retrieved memory may not be the nearest memory to the input in terms of Hamming distance. This is because, in the Hopfield model, the operations inside the network cannot be controlled. In other words, once the

probe is presented to the model, one must wait until a fixed point is reached; this may or may not be the right state. The need to have an asynchronous controller which guides the sequence of update operations to the correct fixed point with the help of local and global statistics was pointed out by Stinson and Kak [Stin88a]. A new neural network model was proposed that uses an asynchronous controller in the feedback loop. Whenever there is more than one neuron to update at any moment, the controller makes a choice considering global characteristics of the memories. While the controller is active, the Hopfield network is idle. Two algorithms for using the global statistics were presented: the algorithm Oracle and the method of handles.

In an early bicameral model, the controller itself is not a neural network, and a conventional computer (i.e., a von Neumann machine) can serve the purpose. This hybrid approach was mentioned earlier in [Rume86]. It was noticed that people do seem to have at least two modes of operation: one rapid, efficient, and subconscious operation and the other slow, serial, and conscious operation. The normal operation of retrieval is controlled by the fast, subconscious part of the brain. On the other hand, the learning of new information is governed by the slow, conscious system. A conventional computer is appropriate for this conscious part, because it might need

global and local information and it has to handle complex learning algorithms.

2. The Hopfield model may have "spurious" memories which are created because of interference among stored memories. Among them, there are complements to the originally stored memories and others which are linear combinations of some fixed points. Consequently, the model can converge to these wrong, stable points.

In the Hopfield model, each stable point, either an original or a spurious memory, forms an *attraction basin* that influences the dynamics of the probe. If the probe takes a wrong direction influenced by the attraction basin of a spurious memory, it might converge to a spurious memory.

In spite of all these problems, the Hopfield model is an attractive starting point for using neural networks. McEliece, et. al. [McEl87] and Abu-Mostafa and Jacques [AbuM85] analyzed the information capacity of Hopfield model using statistical arguments. McEliece, et. al. found that N/(log N) is the upper bound for the number of memories that can be stored in the Hopfield model with the Hebbian rule. Abu-Mostafa and Jacques showed that the number of patterns that one can store in the Hopfield model is bounded above by N. Their definition of the capacity m is that every set of m patterns that one wishes to store has an zero-diagonal weight matrix T such that

each pattern is a fixed point. This estimation was made without regard to the error correction capability of the Hopfield model. The capacity and performance of the Hopfield model with higher-order correlations were investigated by Prados [Prad88a] and Prados and Kak [Prad88c]. It was shown that with higher-order correlation, the Hopfield model can store up to 2^{N-1} states, where N is the number of neurons in the network. It is a significant improvement in terms of capacity. Amit, et. al. extended the Hopfield model to allow the storage and retrieval of biased patterns which demonstrate the level of activity among neurons [Amit87a]. They noticed the fact that pattern recognition usually deals with images in which the background presents a much larger area than the foreground. Such images are considered to have a low level of activity. They also examined the statistical mechanics of neural networks near saturation and the behavior of the Hopfield model with the number of patterns approaching its maximum capacity [Amit87b].

As mentioned before, neural networks are self-adaptive systems which try to reconfigure themselves according to incoming new information by adjusting the synaptic strengths of connections among neurons. This is called a learning process. Therefore, the learning algorithm plays a very important role in neural networks.

Although there are many variations, the following two learning algorithms are important: the Hebbian rule [Hebb49] and the Widrow-Hoff (or the Delta) rule [Widr60]. The basic idea behind the Hebbian rule is as follows: if two neurons are highly active, the synaptic weight between them should be strengthened by "some" amount. The sum of the outer product, which is used in the Hopfield model, is a simple example of the Hebbian rule. The main reason why the capacity of the Hopfield model is so restricted is that as we store more information in the network, the simple Hebbian rule can make previously stored informations disappear. In 1960, Widrow and Hoff [Widr60] proposed a variation of the Hebbian rule, the Delta rule, which significantly improves the capacity of neural networks. The idea is that the amount of change in synaptic strength should be proportional to the difference between the weighted sum computed at a neuron and the value it should have. Prados and Kak investigated the capacity of the Hopfield model using the Delta-rule and concluded that more than N memories could be stored in the network [Prad89]. Kanter and Sompolinsky proposed a nonlocal learning rule that explicitly used the correlations among patterns [Kant87]. They noticed the problem of spurious memories that result from correlations among patterns.

There are two ways of improving the performance of neural networks. One is to improve the learning algorithm. The Delta rule significantly improves the performance over the Hebbian rule, although with an increased cost of implementation. Another new algorithm involving the unlearning of spurious memories forms a major contribution of this dissertation and it will be presented in chapter 3. The other way to improve performance is to control the direction of convergence, as in a bicameral structure. Two control algorithms are presented in the subsequent chapters: the bicameral classifier and the method of iteration.

There are other types of neural networks in the literature such as the Hamming net, Grossberg's perceptron [Gros88] and Kohonen's model [Koho88]. Lippmann [Lipp87] has presented a taxonomy of these neural networks based on the characteristics of input and learning. Another classification can be made on the types of neurons. There are three types of neurons in the neural network. In terms of functionality, each neuron does the same job, which is to receive inputs from its neighboring neurons and compute an output value which is sent out to its neighbors. But when we view the network as a black box, there exist input, output and hidden neurons. Input neurons receive information from the outside, i.e., a probe. Output neurons send the answer to the outside. Hidden neurons are

internal to the network and are invisible to the outside. They work on the internal representations which might be necessary in solving complex problems. Therefore, we can divide the neural networks into two classes: one which employs hidden neurons and one which does not. Hopfield's associative memory model does not have hidden neurons, whereas the Hamming net utilizes them. Put another way, Hopfield's model has potentially full-bidirectional connectivity, whereas models with hidden neurons have no feedback.

A neural network without hidden neurons is described in Figure 1. Note that the number of input neurons can be different from that of output neurons. In this case, the configurations of input and output patterns are different. When we have the same configurations, input and output neurons can be combined and the representation of Figure 2 can serve the same purpose. For an associative memory, the closeness between the input and output patterns is defined as the Hamming distance. But for other problems where the similarity structure of the input and output patterns are different, we need hidden neurons to produce the correct output patterns as shown in Figure 3. The hidden neurons can perform the encoding of internal representation or intermediate computation which is not visible to the outside. Various problems that require hidden neurons are discussed in [Rume86]. They also generalize the Delta-rule

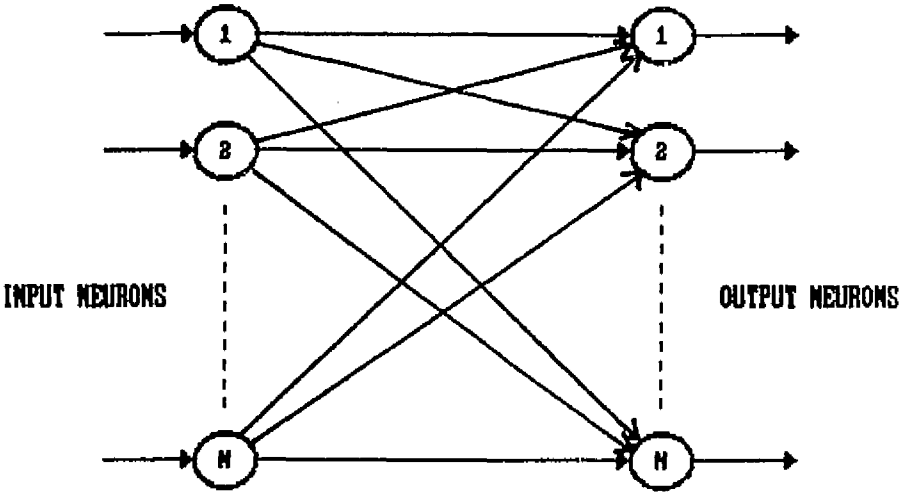


Figure 1

Neural Network without Hidden Neurons

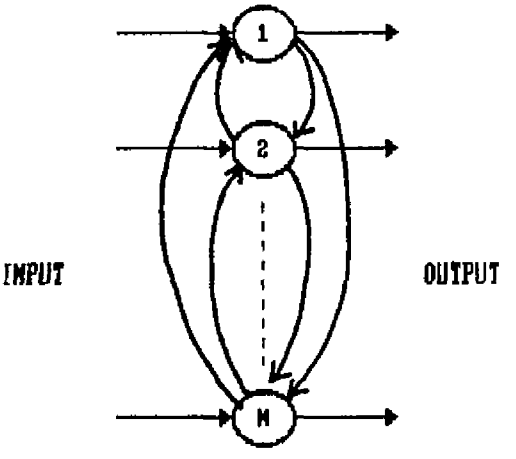


Figure 2

Neural Network without Hidden Neurons

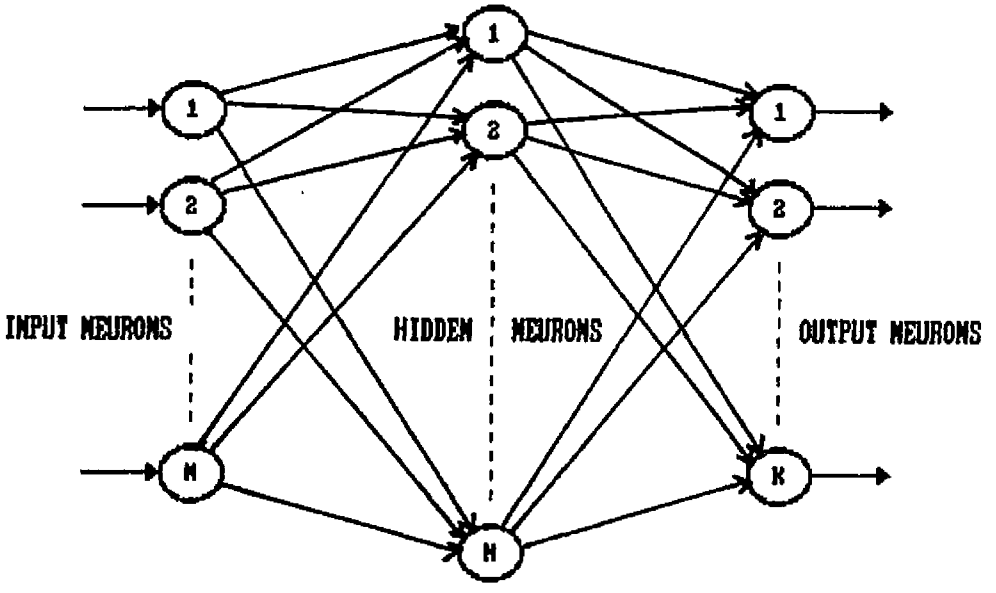


Figure 3

Neural Network with Hidden Neurons

for learning in neural networks with hidden neurons, which is known as the "Backpropagation algorithm".

The Hamming net works on binary inputs and uses the Hamming distance during its operation. It gives a better performance and a smaller number of connections among neurons than the Hopfield model. One problem with this net is that when two states are at the same Hamming distance from the input state, it does not converge to either of them and goes into an infinite loop.

One of the advantages of the Hamming net over the Hopfield model is that it does not suffer from problems of complement and spurious states. Furthermore, it can be modified as an unsupervised pattern classifier using the "leader clustering" algorithm [Hart75]. It has two substructures called *lower subnet* and *maxnet*. The lower subnet computes the total number of components in a pattern minus the Hamming distance to those stored patterns, and passes these distance values to the maxnet. The maxnet selects the largest value among them, which thus represents the closest stored pattern to the input. But this model does not incorporate any learning method. When it receives a new memory, it simply provides an additional number of neurons and makes the necessary connections.

Now consider the architecture of neural networks. Current studies have focused on monolithic structures. It

is well known that the human brain has a "bicameral" nature at the gross level, and several specialized structures. In this dissertation, we investigate the computing characteristics of neural networks that are not monolithic, being enhanced by a controller that can run algorithms that take advantage of the known global characteristics of the stored information. Such networks have been called bicameral neural networks [Stin88a, Stin88b]. Since these networks approach the functioning of the human brain in a certain sense, we expect that our models would be useful in AI problems that require higher level reasoning.

Chapter 2 provides the biological background and implication of the bicameral model and defines the structure and functionality of this model. Bicameral networks are comprised of two separate networks that have different structures and cannot be replaced by a monolithic structure. This requirement is essential because otherwise we could combine them into a monolithic network. It also discusses the Hopfield model and two learning rules: the Hebbian rule and Delta rule.

Chapter 3 presents a new learning algorithm, called the method of Correlation Continuous Unlearning (CCU), that involves the unlearning of spurious memories and has been shown to be superior in performance to the Hebbian rule and the Delta rule. We also discuss the capacity of

various learning rules and how the inclusion of a self-feedback loop affects the capacity of the Hopfield model.

The Stinson and Kak's bicameral model [Stin88a, Stin88b] used asynchronous control. The original model was not exactly a bicameral network, because one of the two subnetworks was not implemented as a neural network. In later work [Stin88b, Kak89a, Kak89b], they did show how information can be indexed in a neural network, which makes it possible for neural networks to implement many standard algorithms like the ones that are run on the synchronous controller. In chapter 4, we present a more advanced bicameral network which can be used for pattern recognition. The network L is the Hopfield model as an associative memory, and network R is an image categorizer based on the Hamming net. The network R processes information at a higher level of abstraction a network L in making decisions regarding which neuron to update. The bicameral classifier is applied to two different models. In one model, it is assumed that every pattern has the same probability of occurrence. The other model represents more realistic situation where patterns can have different probabilities.

Chapter 5 presents a new control algorithm based on the Hamming distance that we call the method of iteration. It has been shown that this method significantly enhances performance.

Chapter 6 draws a comparison between the Hopfield model and multilayered networks. The CCU algorithm has also been applied to Bidirectional Associative Memory (BAM) with greatly improved performance. For multilayered networks, the usefulness of indexing patterns to system performance is examined.

Chapter 7 summarizes the results of the dissertation and suggests areas for future research.

Chapter Two

Background

Early studies of neural network considered either a monolithic feedback structure or hierarchical or layered structures in which information (or signals) flows only in one direction. The visual processing system of the brain is often seen to be layered [Hube62, Hube74]. On the other hand a feedback structure may be viewed as a concatenation of a series of layered structures. The bicameral network is a feedback structure with a controller.

Stinson and Kak [Stin88a] introduce an elementary bicameral structure for improving the convergence of the Hopfield model, but they use a conventional computer to implement several algorithms. In this work, we study advanced bicameral networks which can be used for pattern recognition problems.

2.1. Structure of a Bicameral Network

A bicameral neural network consists of two subnetworks, network L and R as shown in Figure 4. Both networks have different structures and perform different tasks. One of the essential conditions is that these networks can not be replaced by a single network. In one

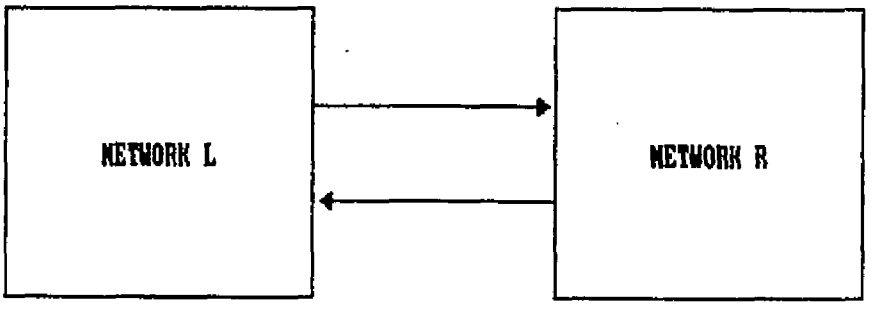


Figure 4

Bicameral Neural Network

of the proposed models, network L is the asynchronous Hopfield model and network R is an image classifier which will assist network L in making decision of choosing a neuron to update.

2.2. The Biological Basis of Neural Networks

One of the main attractions of the neural network approach is the structural simplicity of its basic computational unit. The human brain is known to have many billions of brain cells, and all the intelligent behavior emerges from the interactions of a large number of simple brain cells. The most appealing characteristic of a neural network is its capability of generalization or abstraction. It is believed that human memory and learning seem to rely on the formulation of summary representations that generalize from specific experiences [Rume86]. Also, the error correction capability of the neural network is important. Neural networks are robust and relatively insensitive to missing or erroneous information. Human cognition appears to function well in the event of ambiguity, incompleteness and false information.

The neural network approach was inspired by what we know about the way the human brain works. The basic elements of neural networks are neurons and synaptic interconnections. The human brain is believed to contain

approximately 12 billion neurons and each neuron has about 60,000 dendritic connections [Jack85]. So the human brain is far from being fully connected, whereas in our model as an associative memory, we define the neural network to be so. There are other neural network models which do not require this condition [Rume86, Gros88, Lipp87]. Biologists are still not close to discovering the complete structure and the functionality of the human brain, but many broad characteristics are well-understood. We will first present a brief introduction to neurons and synapses from the perspective of the human brain.

2.2.1. Neurons

The neuron, or nerve cell, has thousands of dendritic connections which receive incoming information from neighboring neurons. It also has axonal branches to transmit information to other neurons. The axon branches out to create synapses on the dendrites of neighboring neurons. The idealized structure of a neuron is drawn in Figure 5. It is not yet completely known how a neuron processes incoming information and transmits an output signal to other neurons, but from the view of neural networks, we usually define the neuron as a thresholding unit characterized by two states, either +1 (firing or active state) or -1 (inactive state). Though we define the

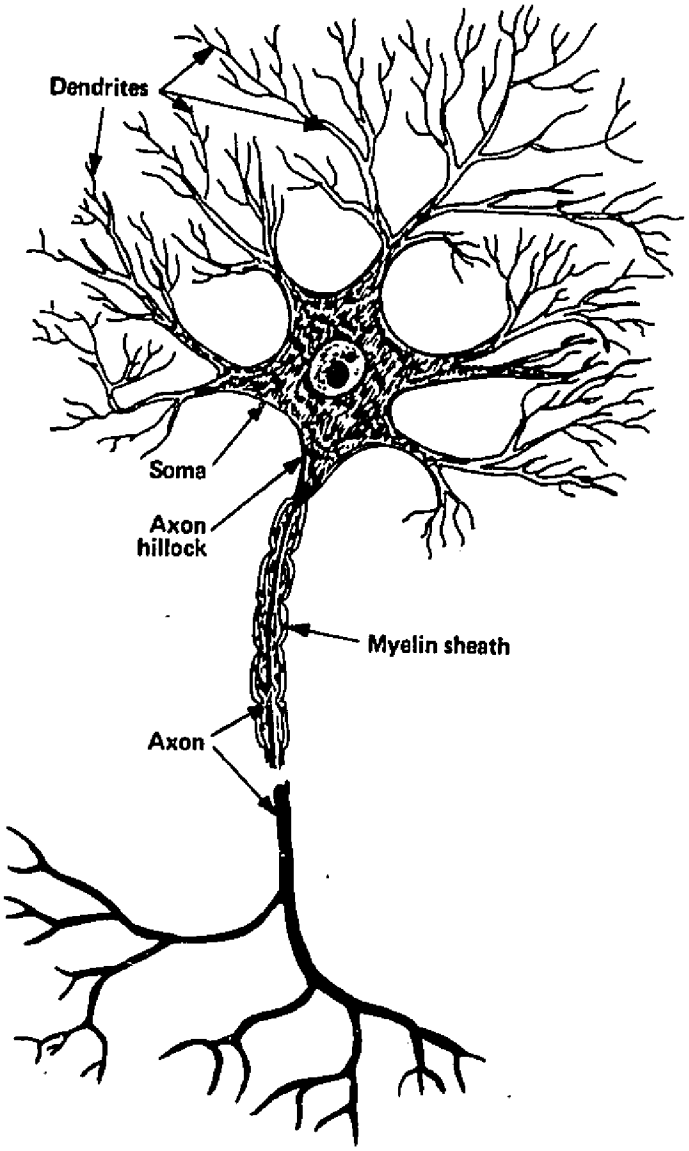


Figure 5

The Idealized Structure of a Neuron

state of a neuron as binary valued, it can be non-binary or continuous.

2.2.2. Synapses

Synapses carry information signals from the axons of a neuron to the dendrites of other neurons. The information transfer across synapses in the cerebral cortex of the human brain is carried out by chemical transmitters. Broadly, there are two types of synapses, excitatory and inhibitory. In neural networks, these types are represented by synaptic strengths. A detailed discussion may be found in [Rume86].

2.2.3. Learning

It is not fully known how humans learn, memorize and forget information, but in neural networks the learning process implies an initialization or modification of synaptic strengths. There are two modes of learning for a neural network: supervised and unsupervised. Supervised learning requires an input and target vector. The input vector is presented to the neural network; its output vector is now compared to the target vector and if there is any discrepancy, the neural network goes into an iterative learning phase again modifying synaptic

strengths until the desired output can be produced. In unsupervised learning, there is no target vector, and the neural network is assumed to learn the input without the help from the outside, self-organizing itself until it produces a consistent output. This unsupervised learning is more difficult, but from a certain point of view it is more realistic, because often in many applications, we do not have target vectors.

2.2.4. Types of Memory in the Human Brain

From the AI point of view, the human brain is the ultimate model as an efficient memory storage and retrieval system. How humans store and retrieve informations is still poorly understood. Psychologists define three types of memory [Jack85]. *Sensory Information Storage* lasts only tenths of a second. It serves to retain fleeting sensory data until the central nervous system processes it. *Short Term Memory* lasts about 30 seconds. We do not model these two types of memory on artificial neural networks, but the information stored in neural networks can be seen as *Long Term Memory*. Humans appear to lose old information after a certain period of time. Artificial neural networks may be trained by different learning algorithms that will be discussed later.

2.2.5. Structure of the Human Brain

It is well-known that at a gross level, the human brain consists of two hemispheres: the left and right brain [Spri81]. Figure 6 shows the symmetric structure of the human brain: the left and right brain and the corpus callosum, which is the communication medium. Despite the symmetric structure, the functionality of the two brains are quite asymmetric. Psychologists have found out that most people are left-dominant for speech. In other words, the left brain is responsible for their capabilities to hear, understand and speak language. The right brain reacts to stimuli and controls emotion and subconscious activities.

Neurologists have found out that the control of the body's basic movement and sensation is evenly divided between the two hemispheres. A lot of research has been done using patients who have brain damage. Also, numerous studies indicate that there are rather discrete areas of each hemisphere for visual, auditory, olfactory, gustatory, and somatic perceptions [Jack85].

2.3. The Hopfield Model

In 1982, Hopfield proposed a neural network model that could be worked asynchronously. Hopfield's model is a

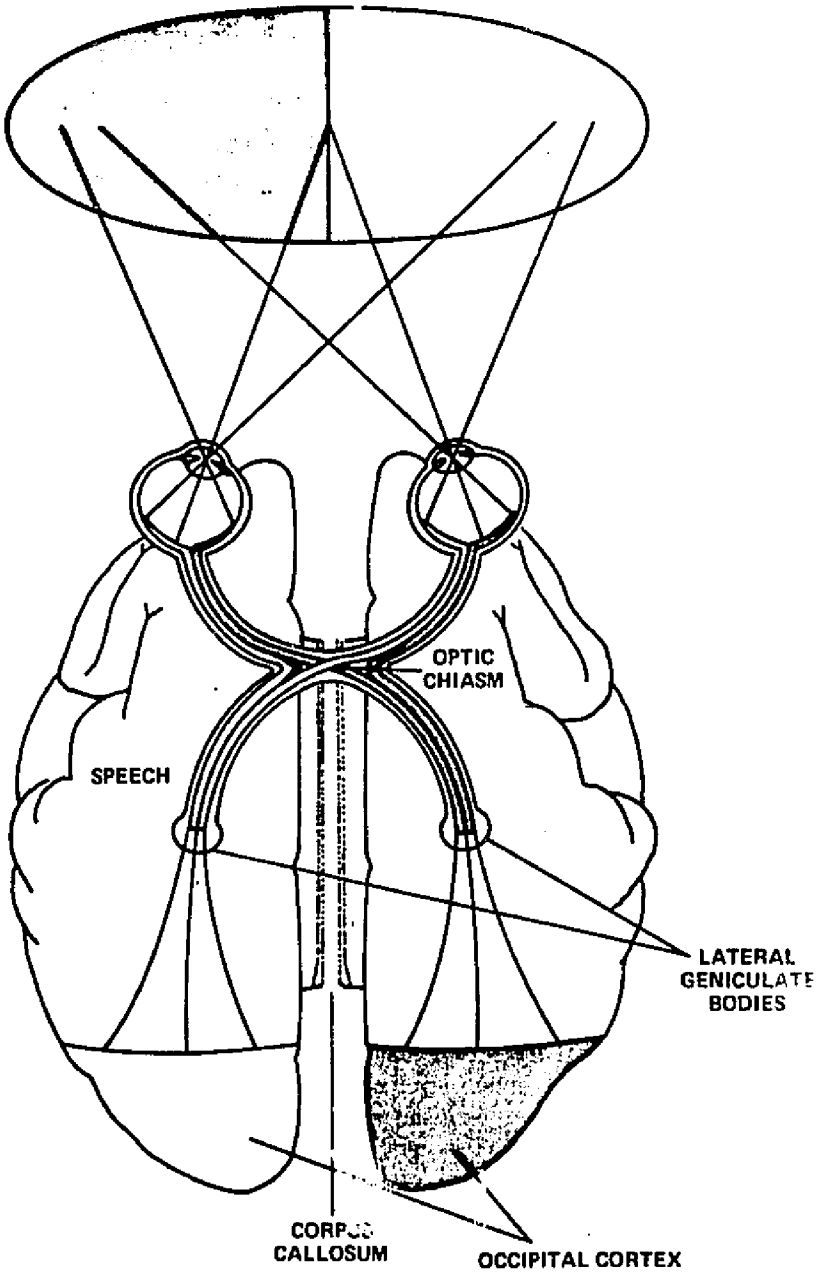


Figure 6

Bicameral Structure of the Human Brain

feedback neural network that serves as a content-addressable memory [Hopf82]. We begin by describing the Hopfield model.

1) It consists of an arbitrary number of neurons, N.

2) Each neuron contains one bit of information. The state of a neuron is either +1 or -1. Each neuron is connected to every other neurons in the network (i.e., completely connected) by arcs which carry weights (or synaptic strengths). These weights, designated T_{ij}, are represented by an N by N matrix, T. The construction of T constitutes a learning algorithm. In his original work, Hopfield used a simple Hebbian rule, which we will discuss later. In this work, we consider two forms of learning, the simple Hebbian rule and a variant of the Delta learning rule.

3) A state of the network is represented by a vector $(x_1, x_2 \text{---}, x_N)$ where x_i denotes the current state of neuron i.

4) The computation takes place at each neuron using a simple thresholding rule, and all neurons have zero as the value of their threshold. The new state x_i' is determined as follows:

$$X_i' = \text{sgn}[\sum_{j=1}^{N} T_{ji} * X_j] = \begin{cases} +1 & \text{if } \sum_{j=1}^{N} T_{ji} * X_j \geq 0 \\ -1 & \text{otherwise.} \end{cases}$$

The new state for the network is determined after all neurons carry out this computation. It can be done either in a synchronous or in an asynchronous manner. If all neurons determine their new values simultaneously, the network is said to have a *synchronous* updating. On the other hand, if each neuron computes its value one at a time, it is an *asynchronous* updating.

It was shown by Hopfield that an asynchronous network guarantees a stable (or fixed) point [Hopf82]. In other words, the network always converges to a stable state. Hopfield showed that for any symmetric weight matrix T, one can always reach a fixed point in the asynchronous Hopfield model [Hopf82]. He proved that for each update, the energy, $-\sum_i \sum_j T_{ij} * X_i * X_j$, does not increase. Similarly it can be proven by showing that the correlation, $C = TX * X = \sum_i \sum_j T_{ij} * X_i * X_j$, is non-decreasing, yet it guarantees strong correlation between the probe and the final answer only when we pick the best x_i to update among several choices. When this is not done, the correlation between x and the final answer might decrease, i.e., $C < 0$, as shown in Figure 7.

2.4. Learning Rules

We begin by storing information in the neural network so as to find responses to probes (or inquiries) later.

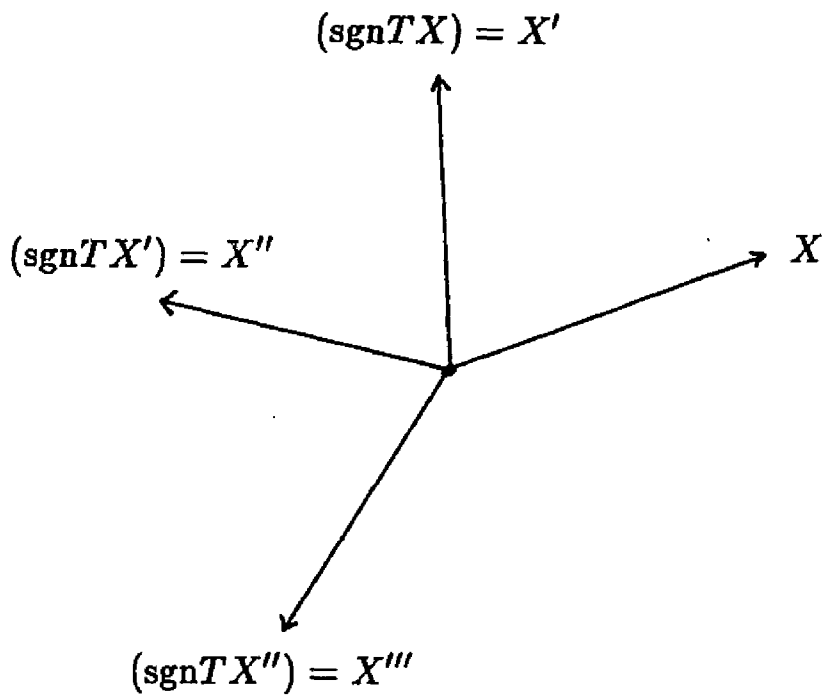


Figure 7

Correlations in Asynchronous Hopfield Model

Since the weight matrix represents all the stored information, although not in the original form, the construction algorithm is crucial to the performance of the network. Several different learning rules have been described in the literature [Rume86, Mins88], but the most important ones are the Hebbian rule and the Delta-rule.

2.4.1. The Hebbian Rule

Hebb described the following simple concept which is basic to many learning rules in his 1949 book [Hebb49]:

> If two neurons are simultaneously excited (or highly active), increase the synaptic strength of the connection between them.

Hopfield used the following simple form of the Hebbian rule of learning [Hopf82]:

Step 1) Initialize matrix T with 0's.
Step 2) Compute the outer product of information vector to store.
Step 3) Add it to the matrix T.
Step 4) Repeat steps 2 and 3 until no more information to store.

The flow diagram of the Hebbian rule is shown in Figure 8. One restriction which Hopfield implemented is that the diagonals of T should be zero ($T_{ii} = 0$ for all i). This means that no neuron has a self-feedback loop. By the nature of this construction, the weight matrix is symmetric. Additional learning can be take place easily, which is not easy in the case of the Delta-rule discussed next. But this simple Hebbian rule used in the Hopfield model has some serious disadvantages.

First, it can produce a wrong answer, i.e., a fixed point which is not the closest one to the probe. Second, it generally contains a complement state to the original memory. Third, it can have spurious states as stable points which are neither original memories nor complements.

Another disadvantage of the Hebbian rule is that if we store too many memories, we might lose some previously stored memories and have spurious memories that we do not intend to store. Because of all these problems, the performance of the Hopfield model is not satisfactory. About 0.15N memories can be stored before the error rate for retrieval becomes severe [McEl87].

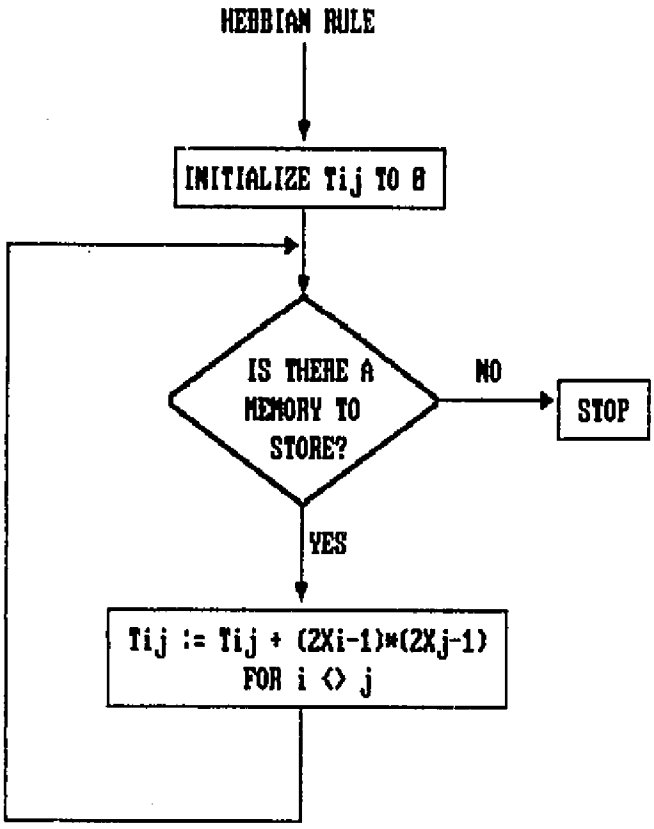


Figure 8

The Hebbian Learning Rule

2.4.2. The Delta Rule

As mentioned before, the Delta-rule is variant of the Hebbian learning rule, and it is often called Widrow-Hoff rule [Widr60]. The basic idea is that the amount of learning, i.e., the amount of change in synaptic strength, should be proportional to the difference between the actual activation value achieved and the target value. The following equation illustrates the idea of the Delta-rule.

$$T_{ij} = L * (t_i(t) - a_i(t)) * o_j(t)$$

where

T_{ij} is the amount of change in the synaptic strength of arc between neuron i and j,

L is the given learning rate ($0 < L < 1$),

$t_i(t)$ is the target activation value of neuron i at time t,

$a_i(t)$ is the actual activation value of neuron i at time t, and

$o_j(t)$ is the output state of neuron j at time t.

Depending on the choice of learning rate and the implementation of **t**, **a** and **o**, one can have many variants of the Delta-rule. Our version of the Delta-rule works as follows:

Step 1) Initialize T with zero's. Set V = [ ].

Step 2) Compute the outer product of the information vector to store and add the information vector to V.

Step 3) Add the outer product to matrix T.

Step 4) Set V' equal to V.

Step 5) If V' is empty, then go to Step 2. Otherwise, pick a state, S, out of V' and run the Hopfield model to get the answer, A.

$$S = (s_1, s_2, ---, s_N)$$

$$A = (a_1, a_2, ---, a_N)$$

If $s_i = a_i$ for all i, then go to Step 5.

Otherwise, set $T_{ij} = T_{ij} + L * (s_i - a_i) * s_j$ for all i, j and go to Step 4.

Figure 9 describes the Delta-rule. Generally, the Delta-rule adjusts the entries of the weight matrix in small steps (or increments), whereas the Hopfield model uses +1 or -1 as the amount of change. The Delta-rule performs better than the simple Hebbian rule because the weight matrix carries only the necessary amount of strength rather than multiples of +1's and -1's. Furthermore, with the simple Hebbian rule, we might erase the previous stored memory when we have too many memories, because interference between memories which are too close to each other can cause problems. Notice that the matrix T

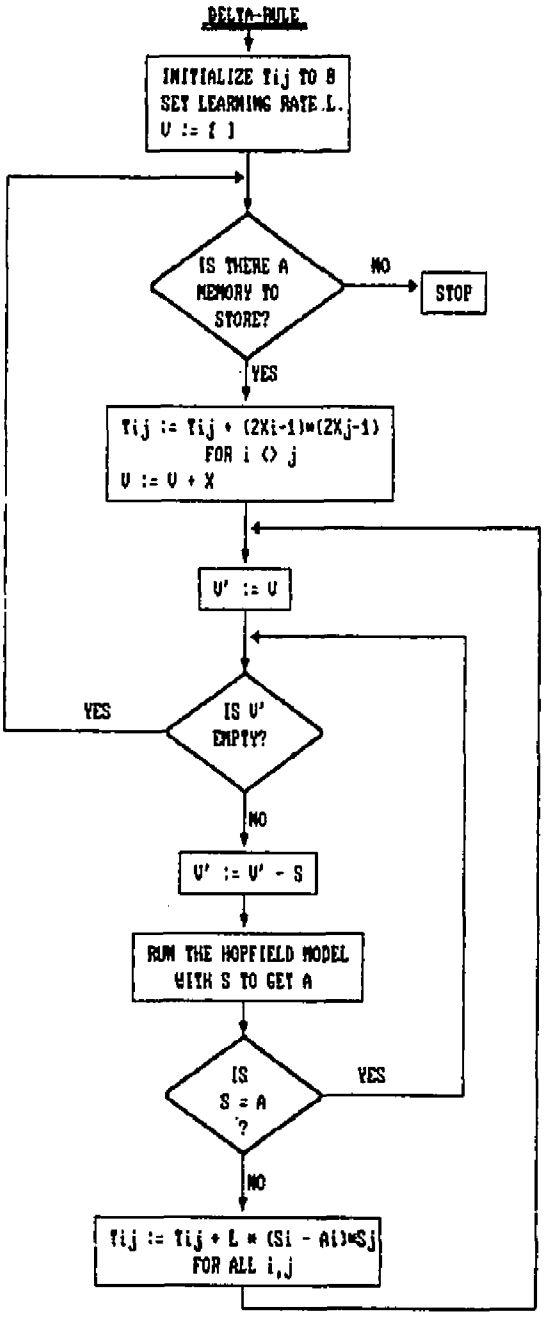


Figure 9

The Delta Learning Rule

of the Delta rule may not be symmetric and can have non-zero diagonals. The Delta-rule significantly improves the performance of the Hopfield model as shown in the tables later, but when the simple Hebbian rule does not erase previously stored memories, the Delta-rule behaves exactly the same as the Hebbian rule. Note that the Delta-rule is constructed to retain all original memories.

One disadvantage of the Delta-rule over the simple Hebbian learning rule used in the original Hopfield network is that we have to keep all stored memories in the controller to check whether they remain as fixed points or not. Whenever we adjust the T_{ij}'s, we take the risk of erasing some of those learned memories; whereas in the Hebbian rule, we don't have to keep previous learned memories. Furthermore, the cost of construction is significantly higher than that for the Hebbian learning rule.

2.4.3. Convergence of the Delta Rule

In their paper [Widr60], Widrow and Hoff introduced an iterative algorithm for solving a set of linear equations. Their idea was originally proposed for the adaptive signal processing problem. Since the neural network is basically a self-adaptive system, we can modify their solution for our model, which is known as the

Widrow-Hoff or Delta rule. The next equation describes their algorithm.

$$T_{ij}(t+1) = T_{ij}(t) + L * (S_i(t) - A_i(t)) * S_j(t)$$

where $A_i(t) = \sum_{j=1}^{N} T_{ij}(t) * S_j(t)$

S is the target vector, and

A is the output vector from the neural network.

Proof of Convergence

Assume that $\varepsilon_i(t)$ is defined as $S_i(t) - A_i(t)$.

Then $\varepsilon_i(t) = S_i(t) - \sum_j^N T_{ij}(t) * S_j(t)$.

If we define $\Delta\varepsilon_i(t)$ as the difference between $\varepsilon_i(t)$ and $\varepsilon_i(t+1)$,

$$\begin{aligned}
\Delta\varepsilon_i(t) &= \varepsilon_i(t+1) - \varepsilon_i(t) \\
&= (S_i(t+1) - S_i(t)) - (A_i(t+1) - A_i(t)) \\
&= -\sum_{j=1}^{N} (T_{ij}(t+1) - T_{ij}(t)) * S_j(t) \\
&\text{since } S_i(t+1) \text{ is equal to } S_i(t) \\
&= -\sum_{j=1}^{N} L * \varepsilon_i(t) * S_j(t) * S_j(t) \\
&= -L * \sum j = 1^N S_j(t)^2 * \varepsilon_i(t)
\end{aligned}$$

Since the term $[-L * \sum_{j=1}^{N} S_j(t)^2] \leq 0$, we can conclude that $\varepsilon_i(t)$ eventually converges to zero. In other words, we can have a stable T matrix which has S as a fixed point.

It is shown that if the input patterns are linearly independent, a solution, i.e., the set of coefficients T, exists. If they are not linearly independent, the algorithm can be modified with the variable learning rate, L(t) = L / t and converge to a solution which is not unique in this case.

This algorithm essentially minimizes the sum of squared errors where the error, $E = S_i(t) - A_i(t)$. In other words, the iterative algorithm will converge to a solution which minimizes $\|S - T * S\|^2$.

Chapter Three

A New Learning Algorithm

There are two ways of improving the performance of neural networks. One is to improve the learning algorithm. Up to now, we have discussed the simple Hebbian rule and the Delta-rule. The Delta-rule performs better than the simple Hebbian rule, although with increased cost of implementation. The other is to control the direction of convergence. The method of iteration and the bicameral model belong to this latter category.

Two algorithms discussed so far deal with learning new information. In this section, we propose a new method of learning which structures the attraction basins differently. This method is based on the concept of "unlearning". This idea of unlearning was noted earlier by Crick, Mitchison and Hopfield [Cric83, Hopf83]. Hopfield's method picks probes at random and weakly unlearns the stable states reached regardless of whether it is an original memory or a spurious state. In other words, this previous method is not tailored to spurious memories. Furthermore, one has to experiment with the number of unlearning events before achieving an optimal result. In a recent dissertation, Potter [Pott87] claimed that one can, with unlearning and self-feedback loops (i.e., T_{ii}'s are

not all zeros), increase the capacity of the Hopfield model up to N, where N is the number of neurons in the network, but his method does not check the spurious memory to unlearn. Our proposed model is based on unlearning the undesirable fixed points of the network, and it does not place any new constraint on the structure. Our method is therefore different from the methods of Hopfield and Potter. We will first describe certain characteristics of fixed points before presenting the new method.

3.1. The Structure Of Fixed Points

Loading a Hopfield model with too many memories can end up in a loss of the previously stored memories and creation of spurious memories that result from interference among stored memories. There are four classes of fixed points in the Hopfield model:

1) original memories
2) complements to original memories
3) non-complement spurious memories
4) complements to fixed points in class 3

When neural networks are "lightly" loaded, it can retain all the original memories. The fact that the Hopfield model operates normally when it has 0.15N

memories can be a measure to determine whether or not the model is lightly loaded. But when the model is "heavily" loaded with many memories, it only has some original memories as fixed points.

3.2. Symmetric Updating

We note that not all class 1 fixed points has its complement in class 2 when we use the following *asymmetric* updating rule:

$$X_i' = \mathrm{sgn}[\sum_{j=1}^{N} T_{ji} * X_j] = \begin{cases} +1 & \text{if } \sum_{j=1}^{N} T_{ji} * X_j \geq 0 \\ -1 & \text{otherwise.} \end{cases}$$

On the other hand, if we use the following *symmetric* updating rule;

$$X_i' = \begin{cases} \mathrm{sgn}[\sum_{j=1}^{N} T_{ji} * X_j] & \text{if } \sum_{j=1}^{N} T_{ji} * X_j \neq 0 \\ X_i & \text{if } \sum_{j=1}^{N} T_{ji} * X_j = 0 \end{cases}$$

all the complements of fixed points in class 1 and 3 will be stable states of the model. This is because x' = sgn [Tx], and for a symmetric updating the complement would be stable for a stable x. Therefore, the number of fixed points in the Hopfield model is even when we use the symmetric updating rule. Also in the asymmetric rule, the

assertion that all complement states are stable states is valid as long as $\sum_j T_{ji} * X_j \neq 0$ for all i.

Example (the Hebbian rule)

Original Memories:

(1,-1, 1,-1, 1,-1, 1,-1, 1)	X_1
(1, 1, 1, 1, 1, 1, 1, 1, 1)	X_2
(-1,-1, 1,-1, 1,-1,-1, 1, 1)	X_3
(1, 1, 1,-1,-1,-1, 1, 1,-1)	X_4
(-1, 1, 1, 1, 1,-1,-1,-1, 1)	X_5

The list of fixed points for this example with the asymmetric updating rule is as follows:

(1, 1, 1, 1, 1, 1, 1, 1, 1)	X_2	Class 1
(-1,-1,-1,-1,-1,-1,-1,-1,-1)	Comp. of X_2	Class 2
(-1, 1,-1, 1,-1, 1 -1, 1,-1)	Comp. of X_3	Class 2
(-1,-1,-1, 1, 1, 1,-1,-1, 1)	Comp. of X_4	Class 2
(-1,-1, 1,-1, 1,-1,-1,-1, 1)	Spurious Mem.	Class 3
(1, 1,-1, 1,-1, 1, 1, 1,-1)	Comp. of above	Class 4
(1,-1, 1,-1, 1,-1, 1, 1, 1)	Spurious Mem.	Class 3

Out of five original memories, X_1,---, X_5, only one memory, X_2, remains stored. The number of fixed points in each class is:

Class 1 ------- 1
Class 2 ------- 3
Class 3 ------- 2
Class 4 ------- 1

For X_1 and X_4, we only have their complements as class 2 fixed points.

With symmetric updating rule, the list becomes

(1,-1, 1,-1, 1,-1, 1,-1, 1)	X_1	Class 1
(-1, 1,-1, 1,-1, 1,-1, 1,-1)	Comp. of X_1	Class 2
(1, 1, 1, 1, 1, 1, 1, 1, 1)	X_2	Class 1
(-1,-1,-1,-1,-1,-1,-1,-1,-1)	Comp. of X_2	Class 2

(1, 1, 1,-1,-1,-1, 1, 1,-1)	X_4	Class 1
(-1,-1,-1,-1,-1,-1,-1,-1,-1)	Comp. of X_4	Class 2
(-1,-1, 1,-1, 1,-1,-1,-1, 1)	Spurious Mem.	Class 3
(1, 1,-1, 1,-1, 1, 1, 1,-1)	Comp. of above	Class 4
(1, 1, 1, 1, 1,-1, 1,-1, 1)	Spurious Mem.	Class 3
(-1,-1,-1,-1,-1, 1,-1, 1,-1)	Comp. of above	Class 4
(-1, 1,-1, 1,-1, 1,-1,-1,-1)	Spurious Mem.	Class 3
(1,-1, 1,-1, 1,-1, 1, 1, 1)	Comp. of above	Class 4

Every stable points has its complement in the list and we have 2 more original memories, X_1 and X_4, than with the asymmetric updating rule. Also, the list for the asymmetric rule is contained in the list for symmetric rule. The number of fixed points in each class is:

Class 1 ------- 3
Class 2 ------- 3
Class 3 ------- 3
Class 4 ------- 3

By forcing the model to keep complements, it retains X_1 and X_4, while under the asymmetric rule, it only retains the complements of X_1 and X_4. The fixed points in Class 2, 3 and 4 are spurious memories. We can divide them into two groups, one for the complements and the other for the rest. For every Class 3 fixed point, we can find a linear combination of some original memories. As an illustration, the spurious memory (-1,-1,1,-1,1,-1,-1,-1, 1) is a linear combination of X_1, X_2 and X_3.

(1,-1, 1,-1, 1,-1, 1,-1, 1)	X_1
(-1,-1,-1,-1,-1,-1,-1,-1,-1)	Comp. of X_2
(-1,-1, 1,-1, 1,-1,-1,-1, 1)	X_3

(-1,-1, 1,-1, 1,-1,-1,-1, 1)	

Furthermore, all complements are linear combinations of some fixed points, because complements are (-1) multiplied by the corresponding fixed point. Due to this property, we can predict the patterns for all spurious memories.

As shown in the above example, the symmetric updating rule increases the number of class 1 fixed points. During our extensive testing, we found out that the symmetric rule performs at least as well as the asymmetric rule.

3.3. The Correlation Continuous Unlearning Algorithm (CCU)

The main reason for the poor performance of the Hopfield model is the correlation (or interference) between the original and the spurious memories. If the memories are too close, the chance of misguiding the direction of a probe is increased. Furthermore the existence of spurious memories in the network does not improve dynamics. They stretch and distort the attraction basins of the memories, which makes it more probable for a probe to take a wrong direction.

The presence of attraction basins around these spurious memories causes serious problems. When we have mutually orthogonal memories, even the simple Hebbian rule performs very well, since the inner-product of any two memories is zero, resulting in zero correlation among memories. The question is how to remove the spurious

memories. As mentioned before, there are two kinds of spurious memories, complements and others which are linear combinations of stable points.

When we make an outer-product of a memory, its complement has the same outer-product. The stable points of Hopfield model are related to the eigenvectors of the weight matrix, T.

sgn [Tx] = sgn [x], where x is the eigenvector.

But at the same time, the following equation also holds.

sgn [T(-x)] = sgn [-x]

Consequently, we cannot remove complements from the Hopfield model. To remove it by subtracting the outer-product from the weight matrix also removes the corresponding original memory. Therefore, we will concentrate on the idea of removing the non-complement spurious memories (class 3) that are encountered while information is being learned.

Basically, this method works similarly to the Delta-rule, except in the process of adjustment. Instead of adjusting entries of the weight matrix as in the Delta-rule, we subtract the outer-product of a spurious memory multiplied by the unlearning rate from the weight matrix as shown in Figure 10.

Step 1) Initialize T with zero's.

Set V = [ ] and specify a value for the unlearning

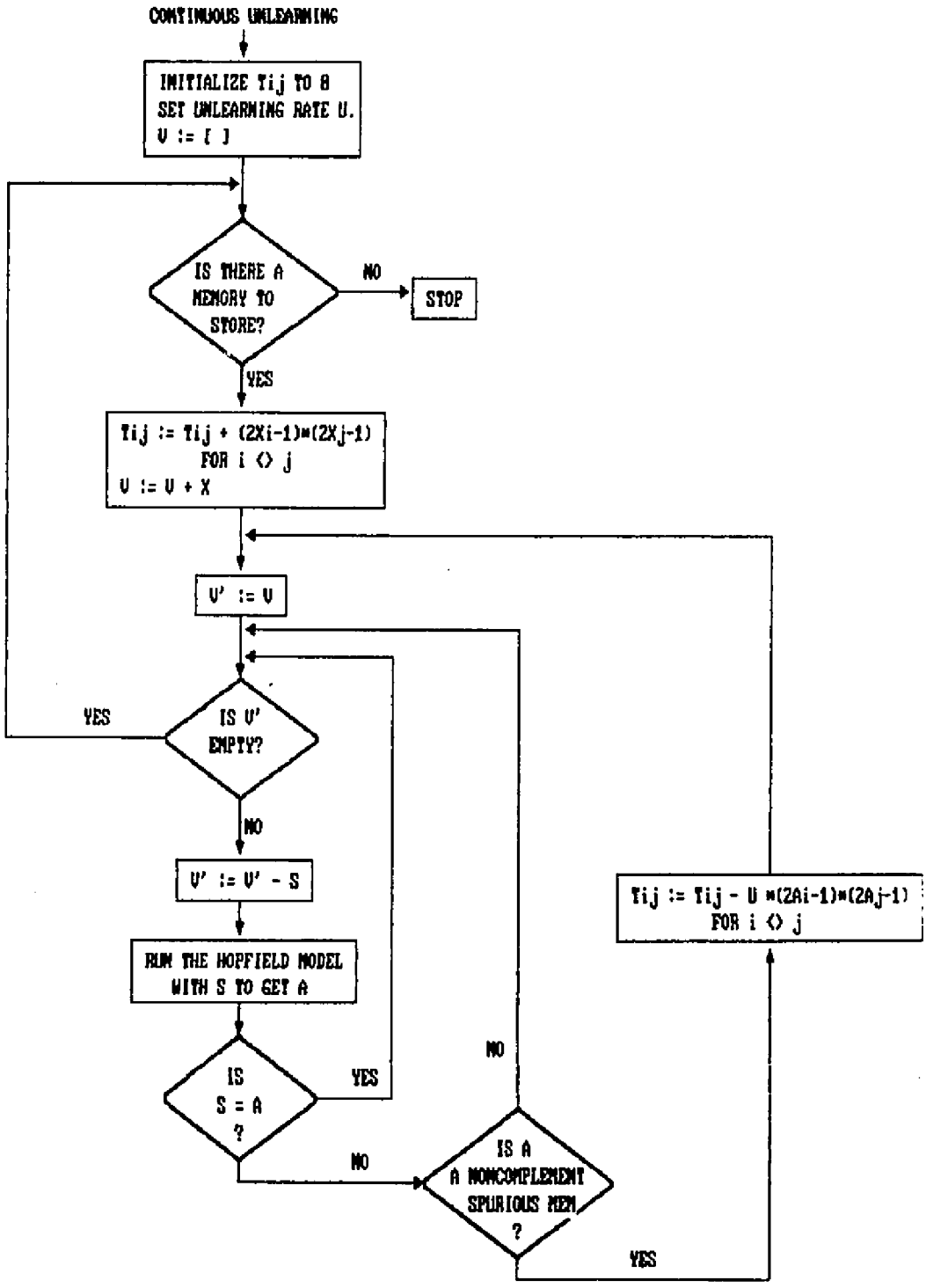


Figure 10

The Correlation Continuous Unlearning Rule

rate, U.

Step 2) Construct the outer-product of a memory to store and add the memory to V. Add the outer-product to T.

Step 3) Set V' equal to V.

Step 4) If V' is empty and there is another memory to store,then go to Step 2. Otherwise, pick a state, S, out of V' and run the Hopfield model to get an answer, A.

$$S = (s_1, s_2, ---, s_N)^t$$

$$A = (a_1, a_2, ---, a_N)^t$$

Step 5) If S = A, then go to Step 4. Otherwise, check to see if A is a spurious memory by comparing it with the original memories. If A is a complement, then go to Step 4. Otherwise, make the outer-product of A, multiply it with the unlearning-rate, U, and subtract it from T. Go to Step 3.

Note that we do not subtract the exact outer-product of a spurious memory from T. Instead, we first multiply it by predefined parameter, U, the "unlearning-rate". The main purpose of this parameter is to adjust the matrix T in small steps to reflect the incremental unlearning.

Another difference between the Delta-rule and the CCU algorithm will now be considered. In step 4, we send a

vector, S, to the Hopfield model to get an answer, A. There are four types of answers we can get.

1) A is the same as S.
2) A is another original memory.
3) A is a complement memory to one of the stored memories.
4) A is a spurious memory which is a linear combination of stable points.

In the Delta-rule, we adjust the entries of T when we have answers of type 2, 3 and 4. But in the CCU, we modify T only when we get answers of type 4.

We provide the comparisons of performance in the following and use the unlearning rate of 0.1. Note that the same examples will be used throughout this work. To get the following performance measures for each example, we test all the possible probes (2^N) and evaluate the characteristics of the answers.

Let C = the number of probes which lead to the correct memory, W = the number of probes which lead to one of the original memories, but not the closest one, and S = the number of probes which lead to spurious memories.

Example 1

```
N = 5
Number of memories = 3
(  1,  1,  1,  1,  1)
(  1, -1, -1,  1, -1)
( -1,  1, -1, -1, -1)
```

	Delta-rule	CCU
C	16	16
W	9	9
S	7	7

Total number of unlearning events = 0

Example 2

```
N = 5
Number of memories = 4
( -1, -1,  1, -1,  1)
(  1, -1, -1,  1,  1)
( -1, -1,  1, -1, -1)
(  1,  1,  1, -1, -1)
```

	Delta-rule	CCU
C	11	15
W	2	5
S	19	12

Total number of unlearning events = 0

Example 3

```
N = 6
Number of memories = 4
(  1,  1, -1, -1,  1,  1)
(  1,  1,  1, -1,  1, -1)
( -1,  1, -1,  1,  1, -1)
( -1, -1, -1, -1, -1, -1)
```

	Delta-rule	CCU
C	19	27
W	5	11
S	40	26

Total number of unlearning events = 4

Example 4

N = 9
Number of memories = 5
(1, -1, 1, -1, 1, -1, 1, -1, 1)
(1, 1, 1, 1, 1, 1, 1, 1, 1)
(-1, -1, 1, -1, 1, -1, -1, 1, 1)
(1, 1, 1, -1, -1, -1, 1, 1, -1)
(-1, 1, 1, 1, 1, -1, -1, -1, 1)

	Delta-rule	CCU
C	120	169
W	51	57
S	341	286

Total number of unlearning events = 14

Example 5

N = 9
Number of memories = 5
(1, 1, 1, 1, 1, 1, -1, 1, 1)
(-1, 1, 1, 1, -1, 1, -1, 1, -1)
(-1, -1, 1, 1, -1, -1, -1, 1, -1)
(1, -1, -1, -1, -1, -1, 1, 1, -1)
(1, -1, -1, 1, -1, -1, 1, -1, -1)

	Delta-rule	CCU
C	135	203
W	39	48
S	338	261

Total number of unlearning events = 12

Example 6

N = 12
Number of memories = 7
(1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 1, -1)
(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
(-1, -1, -1, 1, -1, 1, 1, 1, -1, 1, 1, 1)
(-1, 1, 1, -1, -1, -1, 1, -1, -1, 1, 1, -1)
(-1, -1, -1, -1, 1, 1, 1, -1, 1, 1, 1, 1)
(1, 1, 1, 1, 1, 1, 1, 1, -1, -1, -1, -1)
(1, 1, 1, -1, -1, -1, -1, -1, -1, 1, 1, 1)

	Delta-rule	CCU
C	842	1017
W	571	820
S	2683	2259

Total number of unlearning events = 18

As shown in the table above, the CCU algorithm improves the performance of the Hopfield model. We have seen that unlearning reduces the number of spurious memories which are linear combinations (Class 3 fixed points). The list of fixed points after applying unlearning for example 4 becomes

(1,-1, 1,-1, 1,-1, 1,-1, 1)	X_1	Class 1
(-1, 1,-1, 1,-1, 1,-1, 1,-1)	Comp. of X_1	Class 2
(1, 1, 1, 1, 1, 1, 1, 1, 1)	X_2	Class 1
(-1,-1,-1,-1,-1,-1,-1,-1,-1)	Comp. of X_2	Class 2
(1, 1, 1,-1,-1,-1, 1, 1,-1)	X_4	Class 1
(-1,-1,-1, 1, 1, 1,-1,-1, 1)	Comp. of X_4	Class 2
(1, 1, 1,-1, 1,-1, 1, 1, 1)	Spurious Mem.	Class 3
(-1,-1,-1, 1,-1, 1,-1,-1,-1)	Comp. of above	Class 4

The number of fixed points in each class is:

Class 1 ------- 3
Class 2 ------- 3
Class 3 ------- 1
Class 4 ------- 1

We include the number of unlearning events in the tables because the more unlearning that happens, the better the performance. In this analysis, we should consider the size of the examples. Example 6 has more unlearning than example 4, but its size, 12, is larger

than example 4, which has 9 neurons. But the percentage of improvement for example 4 is larger than example 6.

In our method, we do not have to experiment with the number of unlearning events as in the Hopfield method [Hopf83]. During our extensive testing, we used several unlearning rates. For large U (such as 0.2), some examples entered an infinite loop during the process of adjustment of T. When U is smaller (such as 0.05), the number of unlearning events to converge to a stable T became greater without any noticeable increase in performance. We do not need very small U values (e.g., 0.01), which is the value Hopfield used. For practical purpose, we choose an unlearning rate that is as large as possible without leading to oscillation ("rapid" unlearning). Our method involves some amount of searching through a list of original memories, but with the addition of "orthogonal handles" [Stin88a], we can reduce the amount of searching, because we can determine whether or not an answer is a non-complement spurious memory by looking at its handle.

For our CCU algorithm, we have not found the analytical proof of convergence yet, and the stability of our method has been shown using experimental results. So far we have used the fixed unlearning rate, U, in our extensive testing, but we can consider the use of a variable unlearning rate, $U(t) = U / t$.

One fundamental problem regarding an analytical proof is that we can't express our CCU algorithm in a single mathematical equation, which is possible in the Delta-rule.

$$T_{ij}(t+1) = T_{ij}(t) - U * (2A_i(t) - 1) * (2A_j(t) - 1)$$

where $A_i(t) = \sum_{j=1}^{N} T_{ij}(t) * S_j(t)$

In our method, we have to make sure that A is a non-complement spurious memory. But to determine the type of the answer A, we have to compare the answer with all of the original memories. So we need a program for these comparisons in addition to the mathematical equation. Note that in the Delta-rule, one modifies the T_{ij} whenever $S_i(t)$ is different from $A_i(t)$, which can be expressed by the term $(S_i(t)-A_i(t))$.

We have experimented with larger examples which have more than 15 neurons and provided the results in the next page. Note that our examples are heavily loaded in the sense that they have more than N / 2 stored memories. The numbers in the table represent the total number of probes which produce correct answers. As shown in the table, the CCU algorithm improves the performance about two times more than the Delta rule. But the CCU always results in a larger number of wrong answers than does the Delta rule. For this problem, we can use the bicameral network

N	Number of Memories	Hebbian rule	Delta rule	CCU rule
15	8	3013	3739 (16)	7464 (21)
17	9	10999	19301 (22)	29667 (12)
18	10	731	28639 (31)	45674 (36)
19	10	3754	29343 (35)	68381 (42)
20	11	117118	65637 (42)	131246 (13)

The numbers enclosed in parentheses represent the total number of adjustments made to the matrix T during the learning mode.

approach, which will be discussed in chapter 4, to reduce the number of wrong answers.

There should be two different modes of operation for neural networks. One is the learning mode and the other is the retrieval mode which represents normal operation. Currently the CCU is applied only in the learning mode. One possible extension to this method can be described as follows:

> During the retrieval mode, whenever the model finds a spurious memory, as one can do using "handles", it notifies the controller to resume the learning mode and "unlearn" that spurious memory.

We do not elaborate on this extension at this moment.

3.4. Comparison of Attraction Basins

When an information gets stored in the Hopfield model, it becomes an attractor. The attractor forms a region around it, the attraction basin, where the fixed point attracts the probes in the space formed by the memory. This attraction depends on the closeness among fixed points and the strength of other attractors.

Not all original information sequences become fixed points. Using the Hebbian rule, we might lose some of them and create spurious memories that are undesirable.

Obviously we want the original memories to have large attraction basins. The size or the strength of attraction basin depends on the specific updating rule and the learning algorithm.

Hopfield proved the stability of his model by showing that for each asynchronous update, the energy, $-\sum_i \sum_j T_{ij} * X_i * X_j$, does not increase and thus the probe eventually settles at the minimum, either global or local [Hopf82]. We mentioned the potential problem of the asynchronous Hopfield model in Chapter two.

To investigate the characteristic of the memory space defined in terms of energy, we compare the Hebbian, Delta and CCU learning algorithms using example 3. We begin by ordering all 64 states in such a way that every consecutive state differs by the Hamming distance of 1. Next we compute the energy of each state using the equation $-\sum_i \sum_j T_{ij} * X_i * X_j$.
To compare different learning algorithms, we normalize each value using the smallest entry in the following manner:

	Hebbian	Delta	CCU
(1, 1, 1, -1, 1, 1)	-14.0	-11.6	-8.0

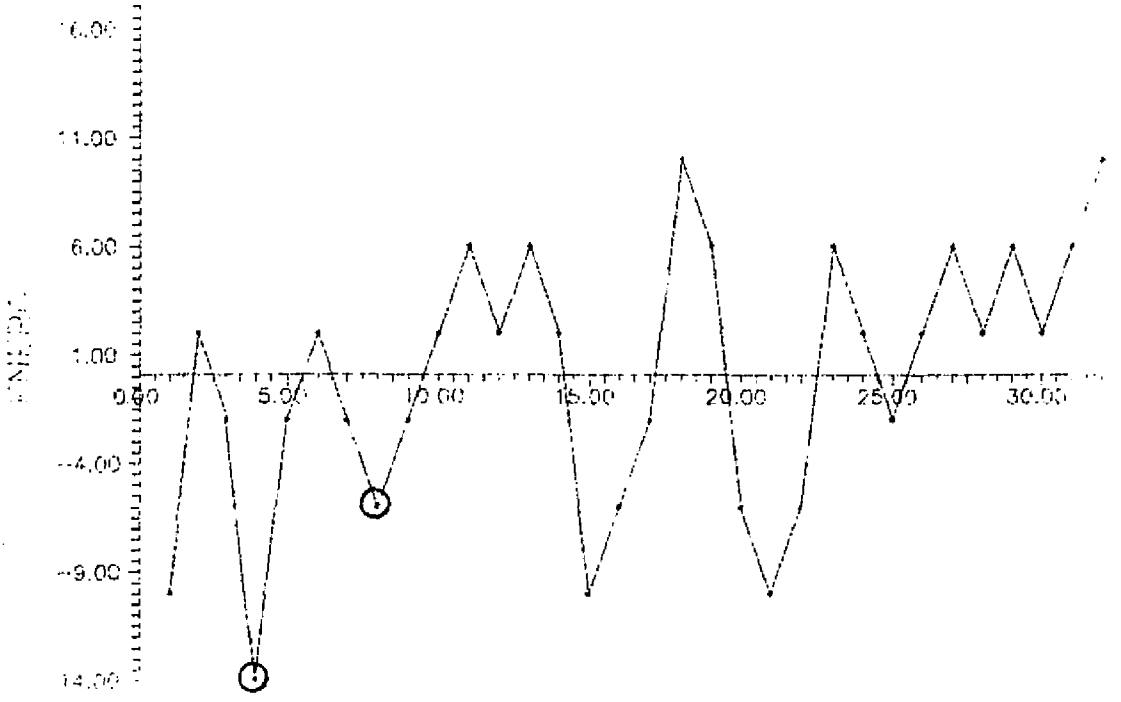
The value x in the Delta rule is normalized by

x * 14 / 11.6

Similar normalization is performed in CCU algorithm too. The results are drawn in Figures 11, 12, 13 and 14. In the figures, we omit all states whose first neuron has the value of -1, because by the nature of the energy equation, every complement state has the same energy as its corresponding original state.

	Hebbian	Delta	CCU
(1, 1, 1, -1, 1, 1)	-14.0	-11.6	-8.0
(-1, -1, -1, 1, -1, -1)	-14.0	-11.6	-8.0

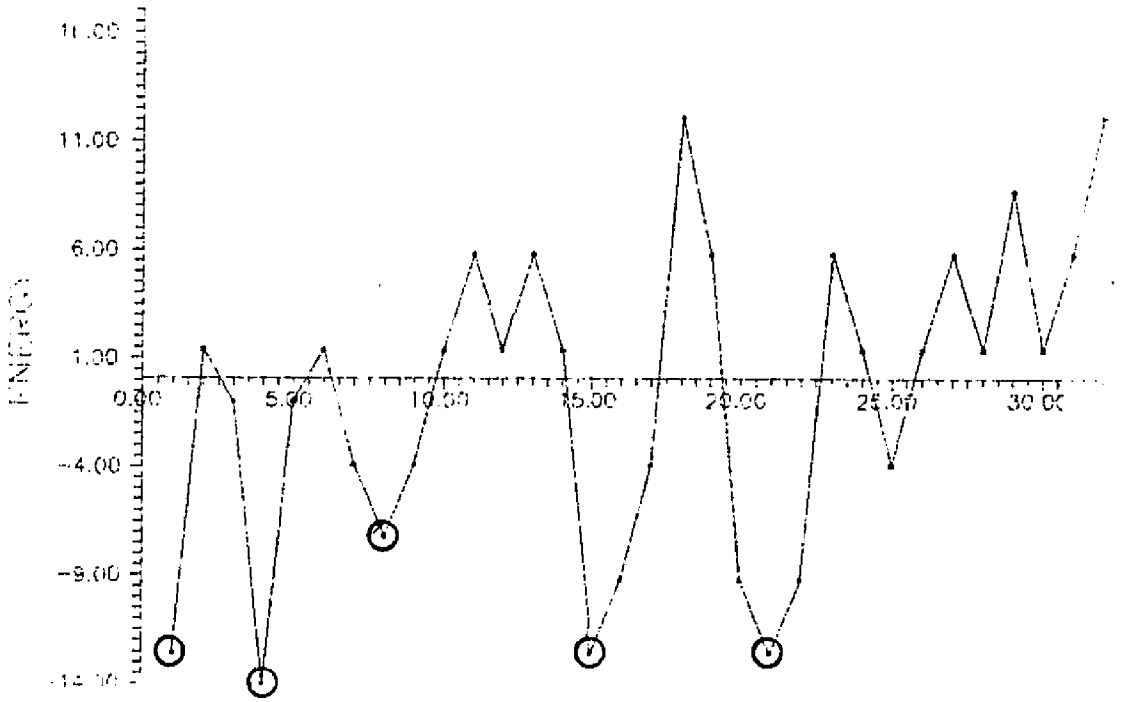
As shown in the figures, all three learning algorithms have similar structures and the figure provides one way of showing the global and local minima in terms of energy. The ordering of states by separating them with the Hamming distance of 1 is not unique. For example, state 6, (1, -1, 1, -1, 1, -1) has the Hamming distance of 1 from state 11, (1, -1, 1, 1, 1, -1), even though consecutive states differ by one bit. Thus, the figures are not expressive enough to show the true structure of state space. The problem comes from the fact that we try to draw the multi-dimensional state space using 2 dimensions. The following is the list of fixed points.



○ dentes the Fixed Point

Figure 11

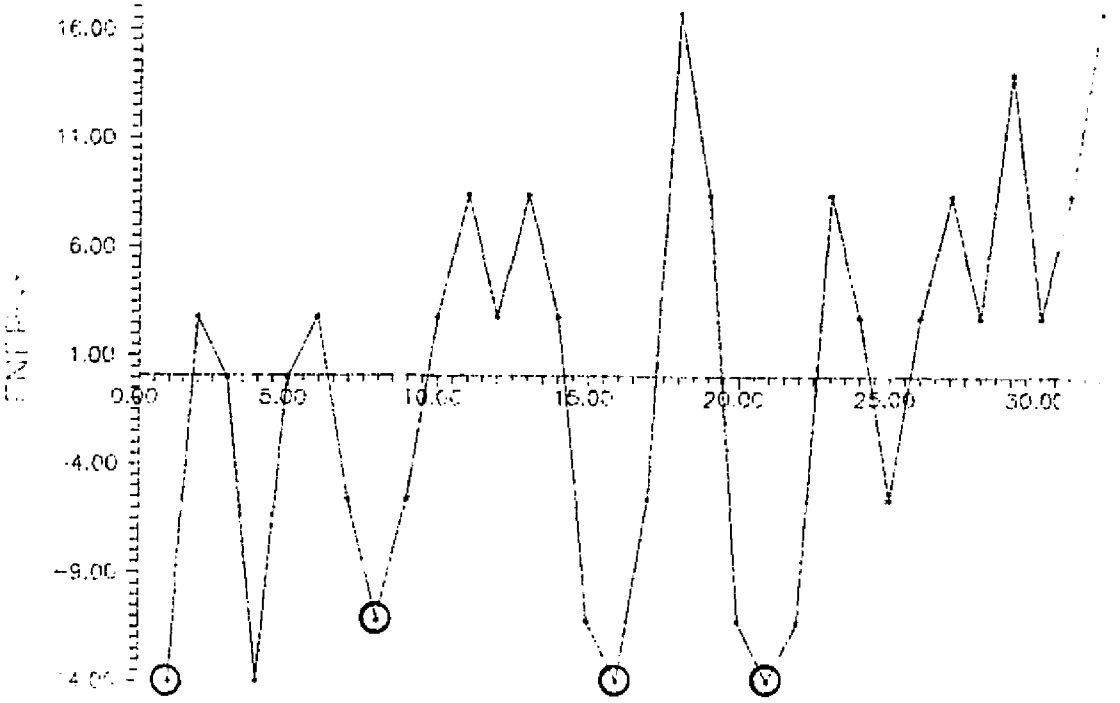
The Energy Structure of the Hebbian Rule



○ dentes the Fixed Point

Figure 12

The Energy Structure of the Delta Rule



○ dentes the Fixed Point

Figure 13

The Energy Structure of the CCU Rule

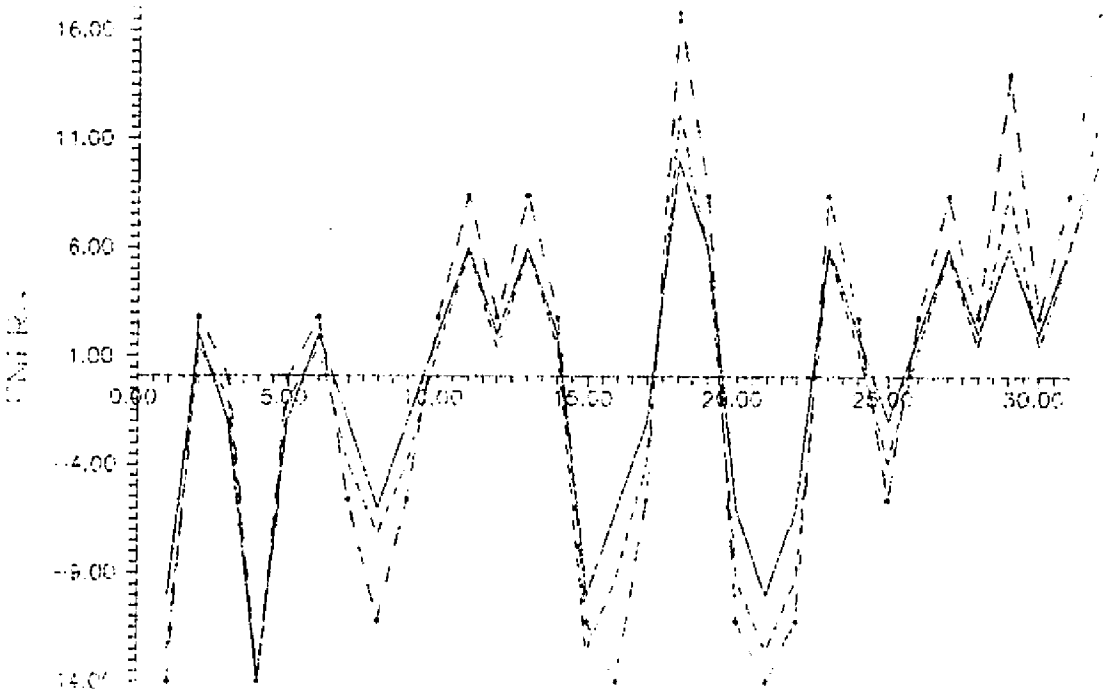


Figure 14

The Energy Structure of Three Learning Rules

Hebbian rule	State Number	
(1, 1, 1, -1, 1, 1)	4	Spurious Mem.
(1, -1, 1, -1, -1, 1)	8	Comp. of Original Mem.

Delta rule		
(1, 1, 1, -1, 1, -1)	1	Original Mem.
(1, 1, 1, -1, 1, 1)	4	Spurious Mem.
(1, -1, 1, -1, -1, 1)	8	Comp. of Original Mem.
(1, 1, 1, 1, 1, 1)	15	Comp. of Original Mem.
(1, 1, -1, -1, 1, 1)	21	Original Mem.

CCU rule		
(1, 1, 1, -1, 1, -1)	1	Original Mem.
(1, -1, 1, -1, -1, 1)	8	Comp. of Original Mem.
(1, 1, 1, 1, 1, 1)	15	Comp. of Original Mem.
(1, 1, -1, -1, 1, 1)	21	Original Mem.

As shown in the figures, the Hebbian rule does not produce a nice structure, because three states (1, 15 & 21) are not fixed points, even though they have less energy than state 8 which is a fixed point. Also it contains only one original memory out of 4. The Delta rule provides a desirable structure, since the states with the 5 smallest energy values remain as fixed points (4 original and 1 spurious memories). The CCU rule is better than the Delta rule in as much as the normalized energy is

less than the Delta rule. In other words, global minima of the CCU are better than those of the Delta rule. Furthermore the CCU does not have the state 4 as a fixed point which is a non-complement spurious memory.

Consider now the energy structure related to the depth of attraction basins. The deeper the basins, the better they are. The width of the attraction basin is an important measure. But the energy depth is not directly related to the width. The number of probes that converge to each fixed point are considered as the width of attraction basin. Since we use random selection for the execution of our example, we run it 50 times and compute the average number of probes. On the next page, we provide the width of attraction basin under the three learning algorithms. It shows that a state and its complement have the same attraction basin in terms of energy and the number of probes. But note that the size of energy does not have any special relationship with the number of probes. In other words, we have to distinguish the depth and width of an attraction basin.

3.5. Comparison of Capacity

So far, we have discussed the performance aspect of the CCU rule; the next question is how many memories can be stored in the network using this method. Several papers

List of Fixed Point	Width of Attraction Basin		
	Hebbian	Delta	CCU
(1, -1, 1, -1, -1, 1)	9.16	9.34	9.22
(-1, 1, -1, 1, 1, -1)*	9.14	9.38	9.18
(1, 1, 1, -1, 1, 1)	23.22	4.18	
(-1, -1, -1, 1, -1, -1)	22.48	3.98	
(1, 1, -1, -1, 1, 1)*		5.9	7.08
(-1, -1, 1, 1, -1, -1)		6.1	7.72
(1, 1, 1, -1, 1, -1)*		5.96	7.08
(-1, -1, -1, 1, -1, 1)		6.66	7.88
(1, 1, 1, 1, 1, 1)		6.32	7.48
(-1, -1, -1, -1, -1, -1)*		6.18	8.36

Fixed Points with "*" represents original memories.

in the literature discuss the question of capacity. Hopfield showed experimentally that about 0.15*N patterns can be stored using the Hebbian rule [Hopf82]. Others have shown that if M memories are chosen at random, the maximum value of M in order that most of the M memories are exactly recoverable is N / (2*log N) using the Hebbian rule [McEl87]. Also Abu-Mostafa showed that the upper bound on the number of memories that can be stored is N without regard to error correction capability of the Hopfield model [AbuM85]. Recently Prados and Kak experimentally showed that more than N memories can be stored in the network using the Delta-rule [Prad88b]. Note that the Delta-rule allows the self-feedback loop, i.e., T_{ii} can be nonzero. But in the CCU rule, we do not have a self-feedback loop. A neural network without a self-feedback loop has one restriction on the set of memories to be stored [Prad88a].

Theorem: For a set of memories to be stored in a neural network without a self-feedback loop, all pairs of memories in the set must differ by more than one bit.

Proof:

Assume that we have the following updating rule.

$$Y_i = \sum_{\substack{i \neq j \\ \& j=1}}^{N} [T_{ij} * X_j] \qquad \text{---------(1)}$$

$$X_i = \begin{cases} +1 & \text{if } Y_i \geq 0 \\ -1 & \text{if } Y_i < 0 \end{cases}$$

Using (1), we know that the next state of the neuron i depends on the states of all the other neurons and the connection weights but not on the state of the neuron i. Assume that two memories differ in only one neuron. In other words,

$$A_j = B_j \text{ for all } i \neq j$$

$A_i \neq B_i$ where A & B are memories to store.

Regardless of the order of presentation to the network, A and B will have the same next state, because neuron i does not affect the outcome. Therefore, if one of two memories becomes a fixed point of the network, the other will reach the same fixed point, and only one of the two can be stored. If we want to store two memories simultaneously in the network without self-feedback, they must differ in more than one neuron.

By similar reasoning, any two memories which are (N - 1) bits apart cannot be stored simultaneously.

3.5.1. Analysis of Capacity

We ran the examples of the size varying from 10 to 20 neurons with the Hebbian, Delta and CCU rules. For each example, we randomly generated 50 sets of memories and tested whether the memories are storable or not. We also counted the number of adjustments made to the weight matrix for the Delta and the CCU rules.

Example: Number of neurons = 12
Number of memories = 6

The Hebbian rule

Memories stored	Number of experiments
0	4
1	10
2	11
3	11
4	7
5	6
6	1

Weighted Average = 2.58

The CCU rule

Memories stored	Number of experiments
0	0
1	0
2	2
3	2
4	3
5	15
6	28

Weighted Average = 5.3
Average Number of Adjustments = 9.0

In the Delta-rule, we could always store the given set of memories. The average number of adjustments is 8.68.

Generally we found out that the CCU rule needs more adjustments than the Delta-rule, especially when we try to store more than N / 2 memories, as shown in the following.

Example: Number of neurons = 17

Number of Memories	Average Number of Adjustments	
	CCU	Delta-rule
7	11.48	11.5
8	23.14	21.16
9	33.28	27.64

In running these examples, we concentrated on the capacity of the network, not on the performance. As shown before, we sometimes ended up storing most of the memories with the CCU rule, whereas the Delta-rule is always successful in storing every memory. But as shown earlier, the performance of the CCU is better than the Delta-rule in terms of correct answers. The main reason behind this fact is that the Delta-rule stores the entire set of memories, but it also creates some spurious memories. On the other hand, the CCU rule is based on the idea of eliminating undesired spurious memories, so the structure of the fixed points is usually better than that of the Delta-rule.

The more memories the network has, the more spurious memories it contains. While the CCU rule tries to remove them, it ends up with more adjustments made to the weight matrix as shown in the table and takes longer to be stabilized. Usually in this situation, the Hebbian rule is able to store very few memories, because when there is much interference among memories, the Hebbian rule loses most of them.

3.5.2. Two Versions of the CCU Algorithm

During the experimentation, we found out that there can be two different versions of the CCU rule. In the Hopfield model, a probe can converge to one of the following four types of memories:

1) One of the original memories, which is the closest to the probe
2) One of the original memories, but not the closest.
3) A complement to an original memory
4) A non-complement spurious memory

In the original version of the CCU rule, we try to unlearn only the type 4 answer. Note that we are not completely removing the non-complement spurious memory at one time, because we use the unlearning rate, U, to

multiply the outer-product and it is usually much smaller than 1. In other words, we are trying to reduce the size of the attraction basin of the spurious memory. Also note that with the CCU rule, we are sometimes unable to store all the given original memories due to the fact that when we get the answers of type 1, 2 and 3, we do not modify the weight matrix. But when we get the answer of type 2, it means that one memory converges to one of the other memories, which is against our purpose, because we want it to converge to itself in order that it becomes a fixed point. The idea now is to reduce the attraction basin of that wrong memory so that the original memory might be able to converge to itself. We can thus make another version of the CCU rule as follows:

Unlearn the answer when we get the types 2 and 4.

As shown in the following table, we found out that this modified version increases the capacity compared to that of the original version.

Number of neurons	Number of Original Memories	CCU	
		Original	Modified
10	5	3.46	4.34
11	6	3.8	4.84
12	7	3.52	5.48
13	8	4.22	6.12
15	7	5.0	6.06
17	9	6.28	7.36
20	10	6.72	7.96

There is one problem with this modified version. It sometimes fails to converge to a stable weight matrix within the given number of iterations (2 or 3 out of 50 experiments), especially when the network has more than N / 2 memories. For those examples, we ran the original version which was always successful in reaching stability. We can therefore combine the two versions in such a way that we mainly use the modified version except when it fails to reach a stable weight matrix, then we use the original version.

It appears that the CCU rule performs best when the network is given about N / 2 memories. In this case, we are successful in storing N / 2 memories most of the time. As the network is saturated with more memories than N / 2, the number of stored memories decreases. But as we stated earlier, the performance of the CCU rule is better than the Delta-rule, even though the Delta-rule is successful in storing all of the given memories.

Chapter Four

The Bicameral Classifiers

So far, we have discussed learning algorithms which deal with the structure of fixed points. Next we propose two control algorithms that affect the direction of convergence: the bicameral classifier, which utilizes the structure of a bicameral neural network. This method is applied to two different models. In one model, it is assumed that every pattern has the same probability of occurrence. The other model represents a more realistic situation where patterns can have different probabilities.

4.1. Models with Equal Probabilities for Patterns

Before we present the bicameral classifier, we introduce the Hamming network [Lipp87] and an image classifier which uses the "sequential leader clustering" algorithm described in [Hart75].

4.1.1. The Hamming Network

Lippmann introduced [Lipp87] a neural network which used the Hamming distance [Hamm82] in measuring the closeness between two memories. Basically, it is a

hierarchical network in which the lower level computes the total number of neurons in a pattern minus the Hamming distance to those stored patterns, and passes these distance values to the upper level. The upper level selects the largest value among them, which thus represents the closest stored pattern to the input.

The Hamming net requires fewer neurons and connections than the Hopfield model. And, it does not suffer from the problem of spurious states that plagues the Hopfield model. But one disadvantage that remains is that if an input is at the same distance from more than one stored memories, the upper level cannot converge and find the answer. Lippmann did not address this problem in his work. There are two possible ways to solve this problem. One is to modify the upper level subnetwork in such a way that if more than one stored patterns are at the same distance from the input, the upper level randomly chooses one of them. It works fine if all the stored memories have an equal probability of occurrence. The other is to assign different probabilities to each of stored memories, and when a tie occurs, one can break the tie using these probabilities. We will discuss this method in section 4.2. On the other hand, the Hamming net does not employ any learning algorithm. When it receives a new pattern, it simply provides an additional number of neurons and makes the necessary connections.

One other relevant aspect is described by the following scenario. There can be a situation where simply finding the closest memory to the input is not enough. The closest stored memory may be useless, because it is too far away from the input. So, we can modify the structure of the lower level in such a way that if the distance between an input and a stored memory is greater than some predefined threshold, we can disable that stored memory so that it won't participate in the operation of the upper level. In the Hopfield model, we cannot control this situation.

4.1.2. The Image Classifier

This classifier implements a clustering algorithm which is similar to the sequential leader clustering algorithm described in [Hart75]. It is built on top of the Hamming network. It takes the first input pattern as the first class. The next input is compared with existing class(es) in terms of the Hamming distance. If the distance is less than a given threshold, it belongs to that class. Otherwise, it constitutes a new class and the number of classes is incremented by 1. The number of classes depends on the value of the threshold and the order of presentation of input patterns.

One disadvantage of this method is that the order of presentation affects the resulting classes. As in the Hamming network, if an input pattern has the same distance to more than one class, it will select one of them in a random fashion, assuming that all classes have an equal probability of occurrence.

In a real-time situation, this assumption might not be valid, because images can have different probabilities or some images can have more importance than others. As an example, we may consider the use of the bicameral neural network in a pattern recognition system in a submarine. Some enemy ships may be more numerous than other ships; also the recognition that it is an enemy is very significant. Certain choices should be given larger weights. We address this issue in section 4.2.

4.1.3. The Bicameral Classifier

This model consists of two neural networks, a Hopfield model and an image classifier built on top of a Hamming network. There are two modes of operation, the learning mode and the retrieval mode.

In the learning mode, the model is initialized according to incoming memories. The Hopfield model will initialize its weight matrix using either the Hebbian or the Delta rule. At the same time, memories are sent to the

classifier, and the classifier determines the number of classes according to a chosen threshold. After classification, the Hamming network is initialized using those resulting classes.

In the retrieval mode, the Hopfield model receives an inquiry (or a probe) and runs until there is more than one neuron to update. When there are several choices, it asks the classifier to make a decision.

But, the Hopfield model and the classifier work on different configurations of a memory. The Hopfield model will have a vector of size N where N, is the number of neurons in the network.

$$X = (x_1, x_2, ---, x_N)$$

The classifier works with two different configurations of a vector. In the learning mode, it works with a vector of size N when it classifies incoming memories. Once it finishes classification, it reduces the size of a vector and initializes the Hamming network with smaller vectors. The reason is that in real-time applications, such as the image recognition problem, the size of an image can be huge. The major contribution of the classifier is to assist the Hopfield model in making decisions. In this case, the response time of the classifier is an important factor. It can be reduced when it works with a smaller configuration, even though the quality of assistance will degrade.

We present two methods to reduce the number of neurons in a memory. Basically, we break up a state into small groups and represent each group by +1 or -1. One way is to compute the decimal equivalent value of a group, and determine the sign of that group. This method is feasible when the number of neurons in a state is not large. But, when it is very large, another simple way is to count the number of +1's and -1's of a group and determine the sign by comparing them. In this case, the number of neurons in a group should be odd.

Example

Suppose that we have a network with 5 neurons and 3 memories.

$$X_1 = (\ 1,\ 1,\ 1,\ 1,\ 1)^t$$
$$X_2 = (\ 1, -1, -1,\ 1, -1)^t$$
$$X_3 = (-1,\ 1, -1, -1, -1)^t$$

When we use the Hebbian rule, the final weight matrix T is as follows:

$$T = \begin{bmatrix} 0 & -1 & 1 & 3 & 1 \\ -1 & 0 & 1 & -1 & 1 \\ 1 & 1 & 0 & 1 & 3 \\ 3 & -1 & 1 & 0 & 1 \\ 1 & 1 & 3 & 1 & 0 \end{bmatrix}$$

At the same time, we send the X_i's to the classifier with the threshold value of 3 and the resulting classes are represented by the following two vectors:

$$C_1 = (1, 1, 1, 1, 1)^t$$

$$C_2 = (-1, 1, -1, -1, -1)^t$$

Notice that X_2 belongs to the first class, C_1, because the Hamming distance between C_1 and X_2 is equal to the threshold of 3.

Since we do not use the same configuration in the Hamming network, we break a state with 5 neurons into 3 groups which have 2 neurons each.

$$C_1 = (1, 1, 1, 1, 1)^t$$

$$C_2 = (-1, 1, -1, -1, -1)^t$$

We compute the equivalent decimal value to each group. For example, (1, 1) becomes 3 ($= 1 * 2^0 + 1 * 2$).

$$C_1' = (3, 3, 1)^t$$

$$C_2' = (-1, -3, -1)^t$$

If we hard-limit the components of these vectors to have either +1 or -1, we have

$$C_1' = (1, 1, 1)^t$$

$$C_2' = (-1, -1, -1)^t$$

Then, we initialize the Hamming network with these C_i's. Let's suppose that we make an inquiry with the probe

$$p = (-1, 1, 1, 1, 1)^t$$

By multiplying p by weight matrix T,

$$\text{sgn}\,[T * p] = p' = \text{sgn}\,[(\ 4,\ 2,\ 4,\ -2,\ 4)^t]$$

Since we hard-limit the value of each neuron using a threshold of 0, we obtain

$$p' = (\ 1,\ 1,\ 1,\ -1,\ 1)^t$$

When we compare p and p', neuron 1 and 4 change their sign. Therefore, we have to select from 1 and 4.

1) Hopfield Random Selection

In the Hopfield model, update choices are random. If it selects neuron 4 to update, the new probe becomes

$$p = (\ -1,\ 1,\ 1,\ -1,\ 1)^t$$

$$\text{sgn}\,[T * p] = p' = \text{sgn}\,[(\ -2,\ 4,\ 2,\ -2,\ 2)^t]$$

Therefore, we have

$$p' = (\ -1,\ 1,\ 1,\ -1,\ 1)^t$$

Since there is no sign change in p and p', we conclude that we reach a stable point (-1, 1, 1, -1, 1). But this state is not one of the original states, X_1, X_2 and X_3. In fact, it is a complement state of X_2.

2) Bicameral Classifier

Consider now the bicameral classifier. Since the Hopfield model has more than one choice, neuron 1 and 4, it asks the classifier for help by sending

$$p = (\ -1,\ 1,\ 1,\ 1,\ 1)^t$$

If we convert p to the smaller configuration, we have

$$C_p = (-1, \ 1, \ 1)^t$$

Then, the Hamming network finds the closest C_i' to C_p in terms of Hamming distance. In this case, it is

$$C_1 = (\ 1, \ 1, \ 1)^t$$

By comparing these two vectors, one can easily conclude that neuron 1 is the better choice than 4, because there is a change of sign in neuron 1. After the classifier returns its recommendation, the Hopfield model resumes its operation. The new probe becomes

$$p = (\ 1, \ 1, \ 1, \ 1, \ 1)^t$$

$$\text{sgn} [T * p] = p' = \text{sgn} [(4, 0, 6, 4, 6)^t]$$

Therefore, $p' = (1, 1, 1, 1, 1)^t$. Since there is no sign-change in p and p', we can conclude that (1, 1, 1, 1, 1) is the stable point, which is the correct answer.

This simple example has shown the advantage of using the bicameral classifier, but the classifier does not always provide optimal results, because it uses classes instead of actual memories. It depends on the value of the threshold used in classifying states and the order of presentation to the classifier. Also, in the Hamming network, when we have two states which are equally distant from the input, we select one randomly.

The issue of assigning different probabilities or weights will be discussed in the next chapter. Generally, the bicameral classifier shows better performance than the

original Hopfield model with random selection. The following table summarizes a comparison of performances using the same examples that are used in chapter 3.

Let C = the number of probes which lead to the correct memory, W = the number of probes which lead to one of the original memories, but not the closest one, and S = the number of probes which lead to spurious memories.

Example 1

	Hebbian rule		Delta-rule	
	Hopfield	Bicameral Classifier	Hopfield	Bicameral Classifier
C	16	23	16	23
W	9	8	9	8
S	7	1	7	1

Example 2

	Hebbian rule		Delta-rule	
	Hopfield	Bicameral Classifier	Hopfield	Bicameral Classifier
C	13	16	11	15
W	4	5	2	1
S	15	11	19	16

Example 3

	Hebbian rule		Delta-rule	
	Hopfield	Bicameral Classifier	Hopfield	Bicameral Classifier

C	7	9	19	23
W	3	7	5	9
S	54	48	40	32

Example 4

	Hebbian rule		Delta-rule	
	Hopfield	Bicameral Classifier	Hopfield	Bicameral Classifier
C	36	43	120	130
W	13	17	51	42
S	463	452	341	340

Example 5

	Hebbian rule		Delta-rule	
	Hopfield	Bicameral Classifier	Hopfield	Bicameral Classifier
C	23	53	135	190
W	4	5	39	34
S	485	454	338	288

Example 6

	Hebbian rule		Delta-rule	
	Hopfield	Bicameral Classifier	Hopfield	Bicameral Classifier
C	243	183	842	1214
W	310	156	571	948
S	3543	3757	2683	1934

As shown in the above tables, the bicameral classifier gives significantly superior performance. The results are always better for the Delta-rule, and are better for the Hebbian rule except when the network is

very heavily loaded as in example 6 because most of the original memories are not stored. In our model, the image classifier is independent of the Hopfield model and it has been implemented using the sequential leader clustering algorithm [Hart75]. More advanced clustering algorithms could also be employed, because the performance of the bicameral classifier depends on the resulting clusters from the clustering techniques.

4.2. Models with Different Probabilities for Patterns

Up to now, we have presented several methods for neural networks in which every pattern has the same probability of occurrence. In this model, if two stored patterns are at the same distance from the input probe, we consider both patterns as correct answers when the input converges to either of the two. Now we consider a more realistic model where patterns have different probabilities.

4.2.1. Pattern Classifiers

Generally, there are two kinds of pattern classifiers: parametric and non-parametric classifiers. The basic assumption behind the parametric classifier is that we know the underlying distributions of the patterns

or memories. Typically, there is the probability of the pattern, $P(X_i)$, and the conditional density function, $p(Y|X_i)$, where X_i is the pattern i and Y is the input pattern to be classified. The optimal classifier can be obtained if we know the $P(X_i)$'s and $p(Y|X_i)$'s. In this case, we can use Bayes' Law to compute the $P(X_i|Y)$'s and select the largest to yield the optimal X_i, but it is hardly a realistic strategy. In most pattern recognition applications, we rarely have complete knowledge about the distribution of patterns. If we have some vague knowledge about the problem, we can use it to select the most applicable known distribution such as the normal or exponential distribution. Next we can estimate the parameters using available sample patterns. Several methods to do this are discussed in [Duda73]. When such knowledge is not available, non-parametric techniques may be used. A comparison of several non-parametric classifiers using linear discriminant functions can be found in [Barn88]. There is the adaptive-clustering neural classifier, the performance of which has been compared with the well-known backpropagation algorithm [Rume86]. But both methods heavily utilize hidden neurons in their method. Also, the number of classes is fairly small compared to the total number of neurons in the network.

Another important factor which can affect the performance of a pattern classifier is whether the

solution space or the set of patterns is linearly separable or not [Duda73]. In this chapter, we consider the simple model in which the a priori probability for each pattern is known, i.e., $P(X_1)$, $P(X_2)$,---, $P(X_M)$ where M is the number of stored patterns, are known.

The new learning and control algorithms presented in the previous chapters do not take these a priori probabilities into consideration. The only applicable method is the bicameral network, in which one network serves as a decision maker, which is the same as in the bicameral classifier presented in section 4.1.3. First, we describe the use of Bayes' Law, which is essential to a statistical approach to the problem of pattern classification [Duda73].

4.2.2. Bayes' Law

The basic assumption of Bayes' Law is that all the relevant probabilities are known, $P(X_i)$ and $P(Y|X_i)$ for all i, where X_i's are stored patterns and Y is the input probe.

$$P(X_i|Y) = \frac{P(Y|X_i) * P(X_i)}{\sum_j P(Y|X_j) * P(X_j)}$$

This theorem shows how to determine an a posteriori probability, $P(X_i|Y)$, from the given a priori probabilities. In the original Bayes' Law, the conditional

probabilities $p(Y|X_i)$ are assumed to be known, but in our model, only the $P(X_i)$'s are given. For our purpose, we substitute $(N - D_i)$ for $p(Y|X_i)$, where N is the number of neurons in the network and D_i is the Hamming distance between Y and X_i. Note that $(N - D_i)$ represents the number of neurons where X_i and Y overlap. Consequently, $P(X_i|Y)$ can be interpreted as the conditional probability of X_i being the correct answer given the input pattern Y. Obviously we choose the X_i that has the largest $P(X_i|Y)$ as an answer.

Suppose we want to build a neural network R, which is the decision maker in the bicameral network, using Bayes' Law. First, we have to consider the number of patterns that will be stored and used in the operation of the network R. If we store all M patterns in network R, we don't need the network L, which is the asynchronous Hopfield model, because the network R alone can function as an associative memory. Note that in the bicameral network, the normal retrieval process is handled by the network L, and the network R will be active only when the network L asks for the decision regarding which neuron to update.

Therefore, we will use one of the existing clustering algorithms to form "superclasses", S_1, S_2,---, S_K $(K < M)$, where K is the number of superclasses for the network R. The clustering algorithm plays the major role in enhancing

the quality of assistance. We can use the following criteria in choosing the clustering algorithm.

1) Algorithms which consider given $P(X_i)$'s in the process of clustering.
2) Algorithms which use some distance measure between two patterns and does not consider $P(X_i)$'s for clustering purpose.

In the "sequential leader clustering" algorithm shown in section 4.1.2, we used the Hamming distance to measure the closeness between two patterns. But we can use other distance measures. In the following, we list several alternatives [Duda73].

Given the clusters C_i, i = 1,---,M, the distance between two clusters C_i and C_j can be

1) $$\min_{X \in C_i, X' \in C_i} \|X - X'\|$$

2) $$\max_{X \in C_i, X' \in C_i} \|X - X'\|$$

3) $$\frac{1}{n_i * n_j} \sum_{X \in C_i} \sum_{X' \in C_j} \|X - X'\|$$

where n_i is the number of patterns in the cluster C_i.

Suppose we have the final clusters, C_1, C_2,---,C_K. Obviously some C_i will have more than one pattern. Then we have to pick one representative S_i out of C_i. One solution is to select the pattern with the largest $P(X_j)$. Also, we have to compute $P(S_i)$. One simple method is to add all $P(X_j)$'s into $P(S_i)$ when the pattern X_j belongs to the cluster C_i. However, we have to make sure that the summation of all S_i is equal to 1.

Second, we will use Bayes' Law to determine the superclass S_i to which the given probe Y belongs. As mentioned before, we substitute $(N - D_i)$ for $P(Y|S_i)$. But the largest $P(S_i)$ will dominate the decision making, shown in the following example.

Example

Suppose the network R has 3 superclasses and we have to decide S_i to which the input Y belongs.

		$P(S_i)$
(1, 1, -1, 1, 1)	S_1	0.5
(-1, -1, -1, -1, 1)	S_2	0.3
(-1, 1, 1, -1, 1)	S_3	0.2
(-1, 1, -1, -1, 1)	Y	

In this example, Y is at the same Hamming distance from S_2 and S_3, but we want S_2 to be the correct

superclass, because $P(S_2)$ is larger than $P(S_3)$. But after computing $P(S_i|Y)$, S_1 is determined as an answer.

$P(S_1|Y)$ = 0.5 * 3 / 3.5 ----> the largest
$P(S_2|Y)$ = 0.3 * 4 / 3.5
$P(S_3|Y)$ = 0.2 * 4 / 3.5

To remedy this situation, we can give more weight to the number of overlapping neurons by squaring it.

$P(S_1|Y)$ = 0.5 * 9 / 12.5
$P(S_2|Y)$ = 0.3 * 16/ 12.5 ----> the largest
$P(S_3|Y)$ = 0.2 * 16/ 12.5

But this simple solution might not work for every problem. Here we have to address one fundamental question: which S_i do we want as an answer for the input Y.

Before squaring the $(N - D_i)$'s, S_1 has the largest probability, but it is not the closest to Y in terms of Hamming distance. We might still want S_1 as an answer depending on the exact nature of the application. After giving more weight to $(N - D_i)$, we can choose S_2 which is the closest to Y and has a larger probability than S_3. We think that the latter method is a better choice generally.

We now provide the performances of three examples used in previous chapters. In running these examples, we used the Delta rule and the same configurations for the networks L and R. Also we cubed the term $(N - D_i)$'s when

we computed the conditional probabilities. When we cluster the memories in the network R, we employ the following measure for the distance between clusters:

$$\max_{X \in C_i, X' \in C_i} \|X - X'\|$$

For each cluster, we select the pattern with the largest probability as the representative for that cluster, sum up all probabilities of patterns in that cluster, and use the sum as the probability for the representative.

Example 4

N = 9	P(X_i)
Number of memories = 5	
(1, -1, 1, -1, 1, -1, 1, -1, 1)	0.25
(1, 1, 1, 1, 1, 1, 1, 1, 1)	0.3
(-1, -1, 1, -1, 1, -1, -1, 1, 1)	0.1
(1, 1, 1, -1, -1, -1, 1, 1, -1)	0.2
(-1, 1, 1, 1, 1, -1, -1, -1, 1)	0.15

Classes in the Network R	$P(S_i)$
(1, -1, 1, -1, 1, -1, 1, -1, 1)	0.5
(1, 1, 1, 1, 1, 1, 1, 1, 1)	0.3
(1, 1, 1, -1, -1, -1, 1, 1, -1)	0.2

	Delta-rule with same prob.	Delta-rule with different prob.
C	120	244
W	51	93
S	341	175

Example 5

N = 9
Number of memories = 5

	$P(X_i)$
(1, 1, 1, 1, 1, 1, -1, 1, 1)	0.2
(-1, 1, 1, 1, -1, 1, -1, 1, -1)	0.1
(-1, -1, 1, 1, -1, -1, -1, 1, -1)	0.4
(1, -1, -1, -1, -1, -1, 1, 1, -1)	0.12
(1, -1, -1, 1, -1, -1, 1, -1, -1)	0.18

Classes in the network R

	$P(S_i)$
(1, 1, 1, 1, 1, 1, -1, 1, 1)	0.2
(-1, -1, 1, 1, -1, -1, -1, 1, -1)	0.5
(1, -1, -1, 1, -1, -1, 1, -1, -1)	0.3

	Delta-rule with same prob.	Delta-rule with different prob.
C	135	311
W	39	50
S	338	151

Example 6

N = 12
Number of memories = 7

	$P(X_i)$
(1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 1, -1)	0.08
(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)	0.1
(-1, -1, -1, 1, -1, 1, 1, 1, -1, 1, 1, 1)	0.11
(-1, 1, 1, -1, -1, -1, 1, -1, -1, 1, 1, -1)	0.12
(-1, -1, -1, -1, 1, 1, 1, -1, 1, 1, 1, 1)	0.14
(1, 1, 1, 1, 1, 1, 1, 1, -1, -1, -1, -1)	0.22
(1, 1, 1, -1, -1, -1, -1, -1, -1, 1, 1, 1)	0.23

Classes in the network R

	$P(S_i)$
(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)	0.18
(-1, -1, -1, -1, 1, 1, 1, -1, 1, 1, 1, 1)	0.25
(1, 1, 1, -1, -1, -1, -1, -1, -1, 1, 1, 1)	0.35
(1, 1, 1, 1, 1, 1, 1, 1, -1, -1, -1, -1)	0.22

	Delta-rule with same prob.	Delta-rule with different prob.
C	842	2355
W	571	1046
S	2683	695

When we compare the performance to that of the model in which every memory has the same probability, the former is much better. We believe that the use of same configuration for both networks is the main reason for this.

Therefore, we have to consider next whether we should use the same configuration for network R as for network L. As mentioned in section 4.1.3, the response time of network R is an important factor. It can be reduced when network R works with a smaller configuration. We presented two simple methods to reduce the size in section 4.1.3.

Chapter Five

The Method of Iteration

We have discussed the bicameral approach to improve the performance of the Hopfield model. We know that the Hopfield model does not work very well, because it can converge to wrong memories and spurious states. To improve the convergence of the Hopfield model, we now propose another method based on the Hamming distance that we call the method of iteration. Later, we apply the concepts of "Hidden-bit" and "Handle" described in [Stin88a], to this new method.

Basically, this method iterates until the Hamming distance between the probe and the response from the Hopfield model is less than or equal to a given threshold. In the Hopfield model, each stable point, either an original memory or a spurious state, forms an "attraction basin" that influences the dynamics of the probe. If the probe takes a wrong direction influenced by the attraction basin of a spurious state, it might converge to a spurious state. The bicameral model tries to guide the probe to take the right direction, although it is not always successful.

So far, when a probe converges to a stable point, all that can be done is to output that stable point as an

answer, hoping that it is the right answer. But that may not happen. We propose that if the response is not close enough to the probe in terms of Hamming distance, several bits should be changed randomly and the Hopfield model runs again with the new probe. Note that during the execution of Hopfield model, update choices will be random. This algorithm is described in Figure 15.

Step 1) Initialize the Hopfield model with original memories. Set the threshold, T, and the number of neurons to change randomly, R.

Step 2) Obtain the probe, P.

$$P = (p_1, p_2, ---, p_N)^t$$

Step 3) Run the Hopfield model to get the answer.

$$A = (a_1, a_2, ---, a_N)^t$$

Step 4) Compute the Hamming distance, D, between P and A. If D <= T, then output A as an answer. Otherwise, select R neurons randomly and change corresponding neurons of A to get P'. Go to Step 3 with P'.

One problem with this method is that it can enter an infinite loop, consisting of steps 3 and 4. Therefore, we must set the limit on the number of iterations to a predetermined value, e.g., 100. Obviously, the performance of this method depends on the choice of threshold, T, and the number of neurons to change, R. If we select too small

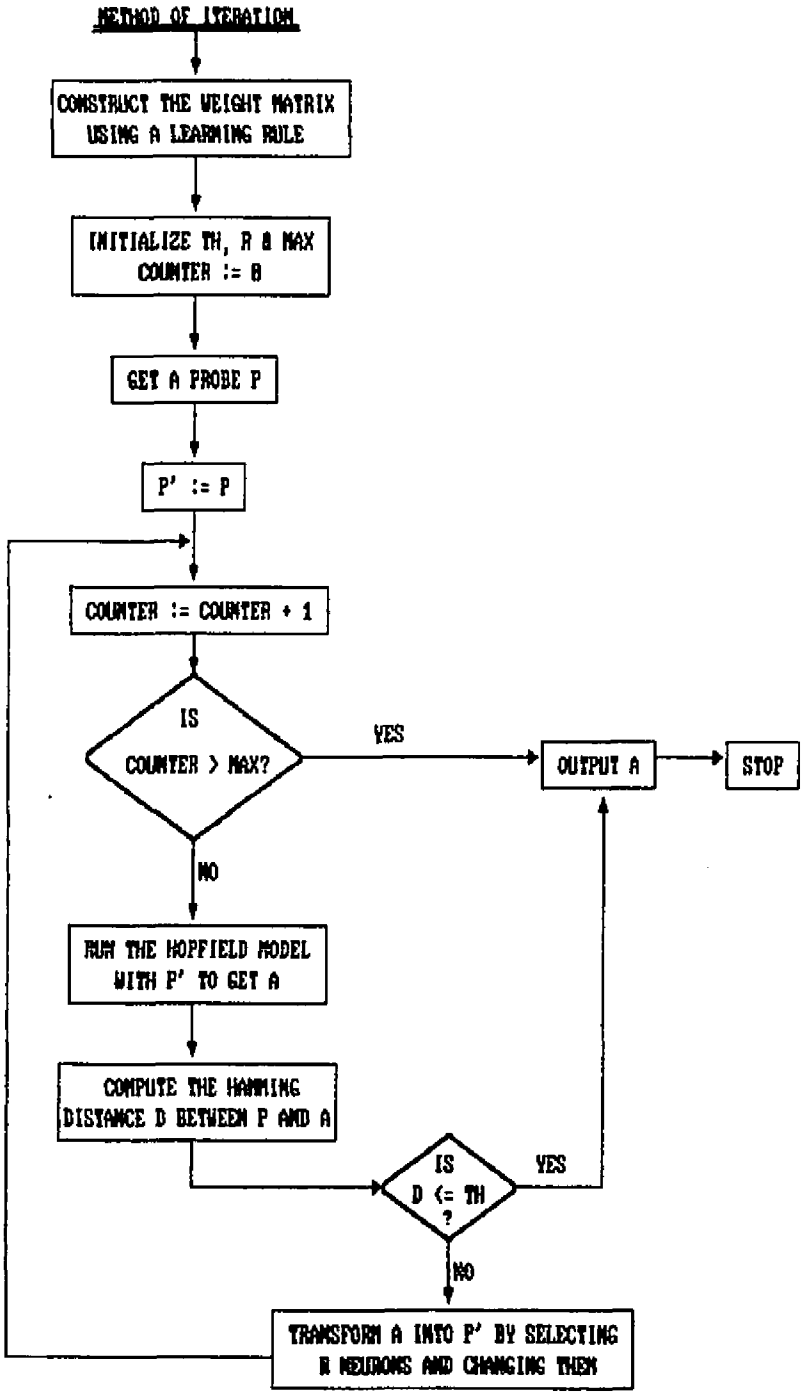


Figure 15

The method of Iteration

a T, this method will have many unnecessary iterations and possibly wrong answers. We find experimentally that T = N/3 and R = N/4 seem reasonable choice for most examples.

In the following, we provide comparisons of the Hopfield model and the method of iteration. Note that the simple Hebbian, the Delta and the CCU rule are used in Hopfield model.

Let C = the number of probes which lead to the correct memory, W = the number of probes which lead to one of the original memories, but not the closest one, and S = the number of probes which lead to spurious memories.

Example 1

	Hebbian rule		Delta-rule		CCU
	Hopfield	Iteration T=2, R=2	Hopfield	Iteration T=2, R=2	Iteration T=2, R=2
C	16	22	16	22	22
W	9	4	9	4	4
S	7	6	7	6	6
Total iterations	32	35	32	35	35

Example 2

	Hebbian rule		Delta-rule		CCU
	Hopfield	Iteration T=2, R=2	Hopfield	Iteration T=2, R=2	Iteration T=2, R=2
C	13	16	11	18	16
W	4	0	2	0	0
S	15	16	19	14	16
Total iterations	32	48	32	40	48

Example 3

	Hebbian rule		Delta-rule		CCU
	Hopfield	Iteration T=2, R=1	Hopfield	Iteration T=2, R=2	Iteration T=2, R=1
C	7	14	19	26	30
W	3	2	5	3	6
S	54	48	40	35	28
Total iterations	64	1356	64	123	115

Example 4

	Hebbian rule		Delta-rule		CCU
	Hopfield	Iteration T=3, R=2	Hopfield	Iteration T=3, R=2	Iteration T=3, R=3
C	36	65	120	160	193
W	13	9	51	20	34
S	463	438	341	332	285
Total iterations	512	8018	512	2453	7168

Example 5

	Hebbian rule		Delta-rule		CCU
	Hopfield	Iteration T=3, R=3	Hopfield	Iteration T=3, R=2	Iteration T=3, R=2
C	23	75	135	173	256
W	4	6	39	8	18
S	485	431	338	331	238
Total iterations	512	12555	512	1665	3867

Example 6

	Hebbian rule		Delta-rule		CCU
	Hopfield	Iteration T=4, R=3	Hopfield	Iteration T=4, R=3	Iteration T=4, R=3

C	243	477	842	1152	1433
W	310	179	571	209	363
S	3543	3440	2683	2735	2300
Total iterations	4096	55792	4096	30371	33990

5.1. The Method of Iteration with a Hidden-bit

Stinson and Kak introduced the concept of a "hidden-bit" to detect complement states [Stin88a]. As seen in previous examples, the number of spurious answers can be quite large. The Hopfield model does not have a way of detecting spurious states, of which there are two kinds:

Complement states, which are complements of original memories, and other spurious states, which are linear combinations of some stable states.

With the addition of one extra hidden-bit, the complement states may be detected. For now, we deal only with the detection of the complement states. Instead of outputting a complement state as an answer, we can notify the user that this answer is considered worthless. Otherwise, the user might consider the answer to be correct.

The detailed procedure of this method is the same as before except in step (1). All the original memories have one extra bit with a value of +1. These memories, of size (N+1), are stored in the Hopfield model. And the probe, with +1 as its hidden-bit value, will be presented to the

Hopfield model. Also, in step (4), if the hidden-bit has +1, the rest of the answer is outputted as the response. Otherwise, the user will be notified that the answer is deemed spurious.

Let C = the number of probes which lead to the correct memory with correct hidden-bit of +1, W = the number of probes which lead to one of the original memories, but not the closest one, SC = the number of probes which lead to spurious memories with hidden-bit of -1, SP = the number of probes which lead to spurious memories with hidden bit of +1, and WK = the number of probes which lead to original memories, but with hidden-bit of -1. Note that "***" implies that this solution is not applicable.

Example 1

N = 5 + 1

	Hebbian rule		Delta-rule		CCU
	Hopfield	Hidden-bit T=2, R=2	Hopfield	Hidden-bit T=2, R=2	Hidden-bit T=2, R=2
C	16	24	16	24	24
W	9	1	9	1	1
SC	***	7	***	7	7
SP	7	0	7	0	0
WK	***	0	***	0	0
Total number of iterations					
	32	93	32	93	93

Example 2

N = 5 + 1

	Hebbian rule		Delta-rule		CCU
	Hopfield	Hidden-bit T=2, R=1	Hopfield	Hidden-bit T=2, R=1	Hidden-bit T=2, R=1
C	13	2	11	19	19
W	4	1	2	0	0
SC	***	6	***	3	3
SP	15	17	19	10	10
WK	***	6	***	0	0
Total number of iterations					
	32	1045	32	36	36

Example 3

N = 6 + 1

	Hebbian rule		Delta-rule		CCU
	Hopfield	Hidden-bit T=2, R=2	Hopfield	Hidden-bit T=2, R=2	Hidden-bit T=2, R=2
C	7	14	19	39	42
W	3	4	5	5	4
SC	***	25	***	14	16
SP	54	21	40	5	
WK	***	0	***	1	1
Total number of iterations					
	64	2107	64	555	559

Example 4

N = 9 + 1

	Hebbian rule		Delta-rule		CCU
	Hopfield	Hidden-bit T=3, R=2	Hopfield	Hidden-bit T=3, R=2	Hidden-bit T=3, R=2

C	36	83	120	208	209
W	13	9	51	17	27
SC	***	163	***	166	175
SP	463	257	341	121	101
WK	***	0	***	0	0

Total number of iterations

512	16951	512	8846	18770

Example 5

N = 9 + 1

	Hebbian rule		Delta-rule		CCU
	Hopfield	Hidden-bit T=3, R=3	Hopfield	Hidden-bit T=3, R=3	Hidden-bit T=3, R=3
C	23	152	135	273	272
W	4	34	39	8	26
SC	***	175	***	113	161
SP	485	151	338	118	50
WK	***	0	***	0	3

Total number of iterations

512	16464	512	7476	14220

Example 6

N = 12 + 1

	Hebbian rule		Delta-rule		CCU
	Hopfield	Hidden-bit T=4, R=3	Hopfield	Hidden-bit T=4, R=3	Hidden-bit T=4, R=3
C	243	924	842	1552	1741
W	310	535	571	295	393
SC	***	1075	***	1200	1254
SP	3543	1562	2683	1049	706
WK	***	0	***	0	3

Total number of iterations

4096	113538	409	75103	86338

Notice that there is a significant reduction in the number of spurious states from the Hopfield model due to the hidden-bit. This is because the complement spurious states are detected and considered as semi-correct answers.

This hidden-bit method has one disadvantage which sometimes shows up in the performance. In example 2, the performance is worse than the original Hopfield model. The reason is that the second original memory, (1,-1, -1, 1, 1), is found to have the largest attraction basin, but with the addition of the hidden-bit of +1, the Hopfield model erases the second state from its memory. In the analysis of the result, we found that many answers are of the form (1, -1,-1, 1, 1,-1), and the last -1, which is the hidden-bit, indicates that it is a complement state. As mentioned before, the simple Hebbian rule of the Hopfield model might erase some of original memories when we try to store too many memories. But as shown in the table, the Delta-rule does not have this problem, because it is constructed to retain all original memories.

5.2. The Method of Iteration with Handles

As mentioned before, the "hidden-bit" method can detect only complement states. Stinson and Kak [Stin88a] proposed the use of "orthogonal handles" to detect other

spurious states. With the addition of orthogonal vectors to original memories, we can detect all spurious states which are linear combinations of stable points. By this orthogonality, every inner-product of any two vectors is zero. If the original memories are mutually orthogonal, even the simple Hebbian rule of the Hopfield model performs very well. For example, the following group of vectors are mutually orthogonal. In other words, the inner-product of every pair of vectors is zero.

```
( 1, 0, 0, 0)
( 0, 1, 0, 0)
( 0, 0, 1, 0)
( 0, 0, 1, 1)
```

If 0's are replaced by -1's, another set of mutually orthogonal vectors is obtained. Now we append orthogonal vectors whose lengths are equal to the total number of memories to be stored. So, the memories of example 1 will be modified as follows:

```
                                !
(  1,  1,  1,  1,  1,  1,!  1, -1, -1)
                                !
(  1, -1, -1,  1, -1, -1,! -1,  1, -1)
                                !
( -1,  1, -1, -1, -1, -1,!  -1, -1,  1)
                                !
  Information part          Handle
```

By this modification, any linear combination of stable points can not have a valid handle, which has only one +1 with the remaining bits being -1's. Therefore, we can detect spurious states.

There are two issues we have to consider.

1) What kind of a handle must we append to the probe before presenting it to the Hopfield model? Our solution is to select one of the above orthogonal handles randomly and append it to the probe.

2) There are two sets of orthogonal handles we can adopt.

```
(  1, -1, -1)        ( -1,  1,  1)
( -1,  1, -1)        (  1, -1,  1)
( -1, -1,  1)        (  1,  1, -1)
```

We tested both sets on the same examples shown before, and found out that there is no noticeable difference in performance. In terms of the threshold, T, and number of neurons to change, R, we should increase them by some amount, because the addition of handles increases the total number of neurons. In the case of orthogonal handles, the Hamming distance between any two handles is 2, so we will use previous T plus 2 as the new value for the threshold. In the following, we provide the comparison of performance.

Let C = the number of probes which lead to the correct memory with correct hidden-bit of +1, W = the number of probes which lead to one of the original memories, but not the closest one, SC = the number of probes which lead to spurious memories with invalid handle, SP = the number of probes which lead to spurious

memories with valid handle, and WH = the number of probes which lead to original memories, but with invalid handle. Note that "***" implies that this solution is not applicable.

Example 1

N = 5 + 3

	Hebbian rule		Delta-rule		CCU
	Hopfield	Handle T=4, R=3	Hopfield	Handle T=4, R=3	Handle T=4, R=3
C	16	16	16	16	16
W	9	3	9	5	5
SC	***	13	***	11	11
SP	7	0	7	0	0
WH	***	0	***	0	0
Total number of iterations					
	32	32	32	32	32

Example 2

N = 5 + 4

	Hebbian rule		Delta-rule		CCU
	Hopfield	Handle T=4, R=3	Hopfield	Handle T=4, R=3	Handle T=4, R=3
C	13	22	11	23	23
W	4	3	2	4	4
SC	***	7	***	5	5
SP	15	0	19	0	0
WH	***	0	***	0	0
Total number of iterations					
	32	66	32	60	60

Example 3

N = 6 + 4

	Hebbian rule		Delta-rule		CCU
	Hopfield	Handle T=4, R=2	Hopfield	Handle T=4, R=3	Handle T=4, R=3
C	7	24	19	21	25
W	3	12	5	4	9
SC	***	28	***	38	29
SP	54	0	40	1	1
WH	***	0	***	0	0
Total number of iterations					
	64	232	64	98	116

Example 4

N = 9 + 5

	Hebbian rule		Delta-rule		CCU
	Hopfield	Handle T=5, R=3	Hopfield	Handle T=5, R=4	Handle T=5, R=4
C	36	171	120	195	204
W	13	72	51	60	52
SC	***	269	***	257	256
SP	463	0	341	0	0
WH	***	0	***	0	0
Total number of iterations					
	512	5558	512	3665	3579

Example 5

N = 9 + 5

	Hebbian rule		Delta-rule		CCU
	Hopfield	Handle T=5, R=4	Hopfield	Handle T=5, R=4	Handle T=5, R=4

C	23	57	135	224	256
W	4	24	39	78	66
SC	***	314	***	210	184
SP	485	0	338	0	6
WH	***	117	***	0	0
Total number of iterations					
	512	9991	512	3877	5105

Example 6

N = 12 + 7

	Hebbian rule		Delta-rule		CCU
	Hopfield	Handle T=6, R=5	Hopfield	Handle T=7, R=4	Handle T=7, R=4
C	243	139	842	1236	1575
W	310	43	571	885	1088
SC	***	1366	***	1168	1427
SP	3543	0	2683	8	5
WH	***	2548	***	799	1
Total number of iterations					
	4096	135841	4096	28403	30551

Notice that a large reduction in the number of spurious states occurs from the Hopfield model and hidden-bit method to the method of handles. But there is a trade-off between these methods. In case of a wrong answer, the hidden-bit method performs better than the method of handle. When a network is heavily loaded, the original memories may not be stored and, therefore, inclusion of handles may erase some memories. This explains the one anomalous result of example 6.

There is one improvement that can be made to the method of iteration with handles. In the previous method, the criterion that stops the iteration is the Hamming distance between the probe and the answer. If it is less than the given threshold, the output is considered as an answer. In the original method, we do not check the answer, whether its handle is valid or not. When it is invalid, we consider the answer as semi-correct. Here we can improve the performance by forcing the model to produce an answer with a valid handle at the cost of an increased number of iterations. So we add one more criterion to the stopping condition: the validity of handle.

As shown before, any valid handle has single +1 with the remaining bits being -1's. During the iteration, if we encounter any answer with an invalid handle, we append a randomly chosen valid handle to the answer and continue until the following two conditions are met:

1) The final answer should have a valid handle.
2) The Hamming distance between the probe and the final answer should be less than the given threshold.

Consequently, we make one modification to the process which transforms the answer into another probe. In the previous method, if the stopping condition is not satisfied, we transform the answer as follows:

Select R neurons of the answer and change them either from +1 to -1 or vice versa.

Note that the neurons which represent the handle can be selected as the R neurons to be changed. But, in this modified method, we divide the answer into two parts, the information part and the handle. We select R neurons only from the information part and change them correspondingly. Then we randomly select one handle out of the set of valid handles and append it to the transformed information part.

As shown in the tables, this minor modification significantly enhances the performance of the previous method. The total number of iterations has increased somewhat, but a large portion of semi-correct answers from the previous method goes to correct answers. But by forcing the network to converge to answers with valid handles, the number of wrong answers has increased.

Example 1

N = 5 + 3

	Hebbian rule	Delta-rule	CCU	
	Handle T=4, R=3	Handle T=4, R=3	Original Handle T=4, R=3	Modified Handle T=4, R=3
C	16	16	16	29
W	3	5	5	3
SC	13	11	11	0
SP	0	0	0	0
WH	0	0	0	0

Total number of iterations

32	32	32	72

Example 2

N = 5 + 4

	Hebbian rule	Delta-rule	CCU	
	Handle T=4, R=3	Handle T=4, R=3	Original Handle T=4, R=3	Modified Handle T=4, R=2
C	22	23	23	27
W	3	4	4	5
SC	7	5	5	0
SP	0	0	0	0
WH	0	0	0	0

Total number of iterations

66	60	60	52

Example 3

N = 6 + 4

	Hebbian rule	Delta-rule	CCU	
	Handle T=4, R=2	Handle T=4, R=3	Original Handle T=4, R=3	Modified Handle T=4, R=3
C	24	21	25	44
W	12	4	9	18
SC	28	38	29	1
SP	0	1	1	0
WH	0	0	0	1

Total number of iterations

232	98	116	385

Example 4

N = 9 + 5

	Hebbian rule	Delta-rule	CCU	
			Original	Modified
	Handle	Handle	Handle	Handle
	T=5, R=3	T=5, R=4	T=5, R=4	T=5, R=4
C	171	195	204	340
W	72	60	52	137
SC	269	257	256	35
SP	0	0	0	0
WH	0	0	0	0
Total number of iterations				
	5558	3665	3579	9263

Example 5

N = 9 + 5

	Hebbian rule	Delta-rule	CCU	
			Original	Modified
	Handle	Handle	Handle	Handle
	T=5, R=4	T=5, R=4	T=5, R=4	T=5, R=4
C	57	224	256	360
W	24	78	66	131
SC	314	210	184	21
SP	0	0	6	0
WH	117	0	0	0
Total number of iterations				
	9991	3877	5105	7640

Example 6

N = 12 + 7

	Hebbian rule	Delta-rule	CCU	
			Original	Modified
	Handle	Handle	Handle	Handle
	T=6, R=5	T=7, R=4	T=7, R=4	T=6, R=5

C	139	1236	1575	2175
W	43	885	1088	1441
SC	1366	1168	1427	313
SP	0	8	5	0
WH	2548	799	1	167

Total number of iterations

135841	28403	30551	130214

Chapter Six

Multilayered Neural Networks

We have discussed several methods that can be applied to networks without hidden neurons shown in Figures 1 and 2. In this chapter, we apply the CCU algorithm to the two-layer bidirectional associative memory and discuss the indexing scheme on multi-layered feedforward networks.

6.1. Bidirectional Associative Memory (BAM)

In the Hopfield model, one usually associates a pattern with itself in an *autoassociative* fashion [Koho84]. When the patterns to be associated are different, it is called *hetero-associative*.

Kosko introduced a bidirectional two-layer feedback neural network that can perform hetero-association [Kosk87, Kosk88]. The structure of the BAM is described in Figure 16. A state of the BAM is defined as (X,Y) where X and Y are patterns to be associated which can be of different size.

In this model, there are two weight matrices, T_f for forward and T_b for backward information flow. The hebbian rule was used for the construction of weight matrices.

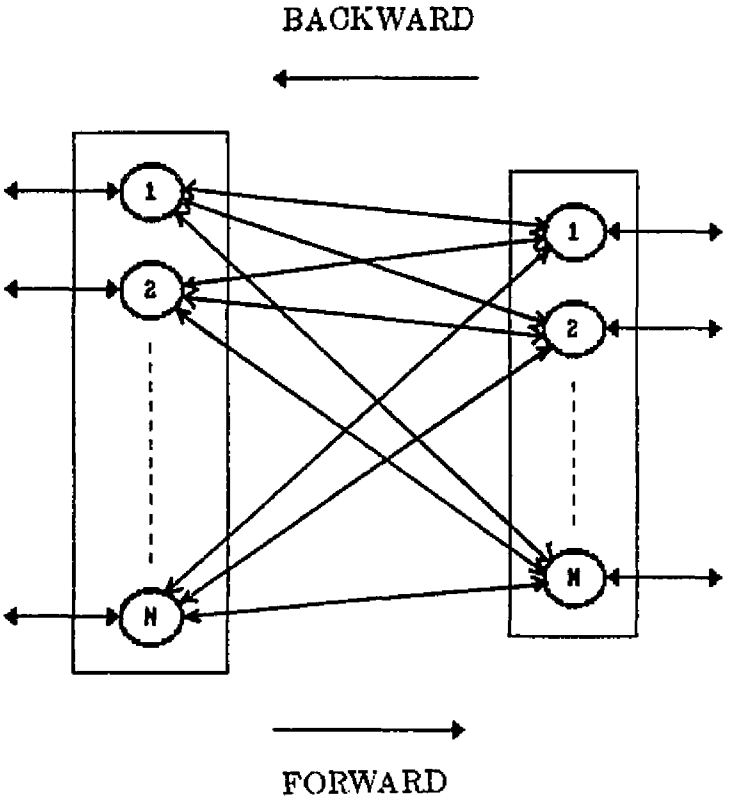


Figure 16

Topology of the Bidirectional Associative Memory

$$T_f = \sum_i X_i^t * Y_i$$

$$T_b = T_f^t = \sum_i Y_i^t * X_i$$

A symmetric updating rule was used. When an input pattern is presented, the system iterates back and forth until it produces a stable state (X_i,Y_i) on both sides. This is called bidirectional stability. Kosko showed that the model always reaches a stable state in both synchronous and asynchronous updates. Synchronous updating requires fewer iterations to reach a stable state than an asynchronous updating, because the change in energy is greater. The Hopfield model can be viewed as a special case of the BAM when X_i and Y_i are same and the diagonal terms are zero.

The capacity of a BAM is the smaller of N and M, where N and M represent the number of neurons in X_i and Y_i respectively. To test the model under a heavily loaded condition, we ran some examples with a number of patterns that exceeds the capacity limit. Also, we adopted the synchronous updating for the example.

Example:

Number of Patterns = 5

List of Original Patterns

(1, -1, 1, -1, 1, -1)	(1, -1, 1, -1)	P_1
(1, 1, -1, -1, 1, 1)	(1, -1, -1, 1)	P_2
(1, 1, 1, -1, -1, -1)	(-1, -1, -1, -1)	P_3
(-1, -1, -1, 1, -1, -1)	(1, 1, -1, -1)	P_4
(-1, -1, 1, -1, -1, 1)	(-1, -1, -1, 1)	P_5

List of Stable States

(1, -1, 1, -1, 1, -1)	(1, -1, 1, -1)	P_1
(1, 1, -1, -1, 1, 1)	(1, -1, -1, 1)	P_2
(1, 1, 1, -1, -1, -1)	(-1, -1, -1, -1)	P_3
(-1, -1, -1, 1, -1, -1)	(1, 1, -1, -1)	P_4
(-1, 1, -1, 1, -1, 1)	(-1, 1, -1, 1)	Comp. of P_1
(-1, -1, 1, 1, -1, -1)	(-1, 1, 1, -1)	Comp. of P_2
(-1, -1, -1, 1, 1, 1)	(1, 1, 1, 1)	Comp. of P_3
(1, 1, 1, -1, 1, 1)	(-1, -1, 1, 1)	Comp. of P_4

(1, -1, -1, -1, -1, -1)	(1, -1, -1, -1)
(-1, 1, 1, 1, 1, 1)	(-1, 1, 1, 1)
(1, 1, 1, -1, -1, 1)	(-1, -1, -1, 1)
(-1, -1, -1, 1, 1, -1)	(1, 1, 1, -1)
(1, -1, -1, -1, 1, 1)	(1, -1, 1, 1)
(-1, 1, 1, 1, -1, -1)	(-1, 1, -1, -1)
(1, -1, -1, 1, 1, -1)	(1, 1, 1, -1)
(-1, 1, 1, -1, -1, 1)	(-1, -1, -1, 1)
(-1, -1, -1, 1, -1, 1)	(1, 1, -1, 1)
(1, 1, 1, -1, 1, -1)	(-1, -1, 1, -1)
(1, -1, 1, -1, 1, 1)	(1, -1, 1, 1)
(-1, 1, -1, 1, -1, -1)	(-1, 1, -1, -1)
(-1, -1, 1, 1, 1, 1)	(-1, 1, 1, 1)

Note that the Hebbian rule fails to store all 5 patterns. Among 21 stable states, 17 patterns are spurious states which cause poor performance. Furthermore, 13 patterns are non-complement spurious states. Therefore we apply the CCU algorithm in the BAM to remove these spurious states.

6.2. CCU with Bidirectional Associative Memory (CCUBAM)

The basic idea is the same as the one given in chapter 3, except that the modification of the weight matrix uses the following equation:

$$T_f = T_f - U * A^t B$$

where (A,B) is a spurious pattern.

We use the same example for this experiment.

Example:
Unlearning rate = 0.01
Number of Patterns = 5

List of Stable States

(1, -1, 1, -1, 1, -1)	(1, -1, 1, -1)	P_1
(1, 1, -1, -1, 1, 1)	(1, -1, -1, 1)	P_2
(1, 1, 1, -1, -1, -1)	(-1, -1, -1, -1)	P_3
(-1, -1, -1, 1, -1, -1)	(1, 1, -1, -1)	P_4
(-1, 1, -1, 1, -1, 1)	(-1, 1, -1, 1)	Comp. of P_1
(-1, -1, 1, 1, -1, -1)	(-1, 1, 1, -1)	Comp. of P_2
(-1, -1, -1, 1, 1, 1)	(1, 1, 1, 1)	Comp. of P_3
(1, 1, 1, -1, 1, 1)	(-1, -1, 1, 1)	Comp. of P_4
(1, 1, -1, -1, -1, -1)	(1, -1, -1, -1)	
(-1, -1, 1, 1, 1, 1)	(-1, 1, 1, 1)	
(1, -1, 1, -1, -1, -1)	(-1, -1, 1, -1)	
(-1, 1, -1, 1, 1, 1)	(1, 1, -1, 1)	

Notice that we have only 4 non-complement spurious states, which, in turn, improves performance. Also the CCU rule reduces the width of attraction basins around spurious patterns if it fails to remove them. In terms of performance, the comparison is as follows:

	BAM	CCUBAM
C	16	23
W	3	6
SP	45	35

We have run several examples and summarizes the improvements in Figure 17. We have shown that the CCU rule significantly improves the convergence. But similarly as in the Hopfield model, the CCU rule increases the number of wrong answers somewhat. We speculate that it can be reduced when we use a bicameral classifier, the Hidden-bit method, and the method of Handles in conjunction with the CCU rule.

6.3. Multilayered Feedforward Neural Networks

We have discussed several methods that can be applied to networks without hidden neurons. In this section, we investigate the feasibility of multi-layered neural networks as an associative memory.

Hecht-Nielsen pointed out that Kolmogorov's theorem [Kolm57] proves the existence of a layered neural network that can map arbitrary input-output pattern pairs [Hech88]. Formally, for any continuous function $\phi : I^n \longrightarrow R^m$,where I is the closed unit interval [0,1] and R is real, ϕ can be implemented exactly by a three layer neural network having n neurons in the input layer, (2n+1) hidden neurons in the middle and m neurons in the output layer. Unfortunately this is only a theorem of existence, and it does not present a constructive method to obtain such networks. However, we do have an algorithm

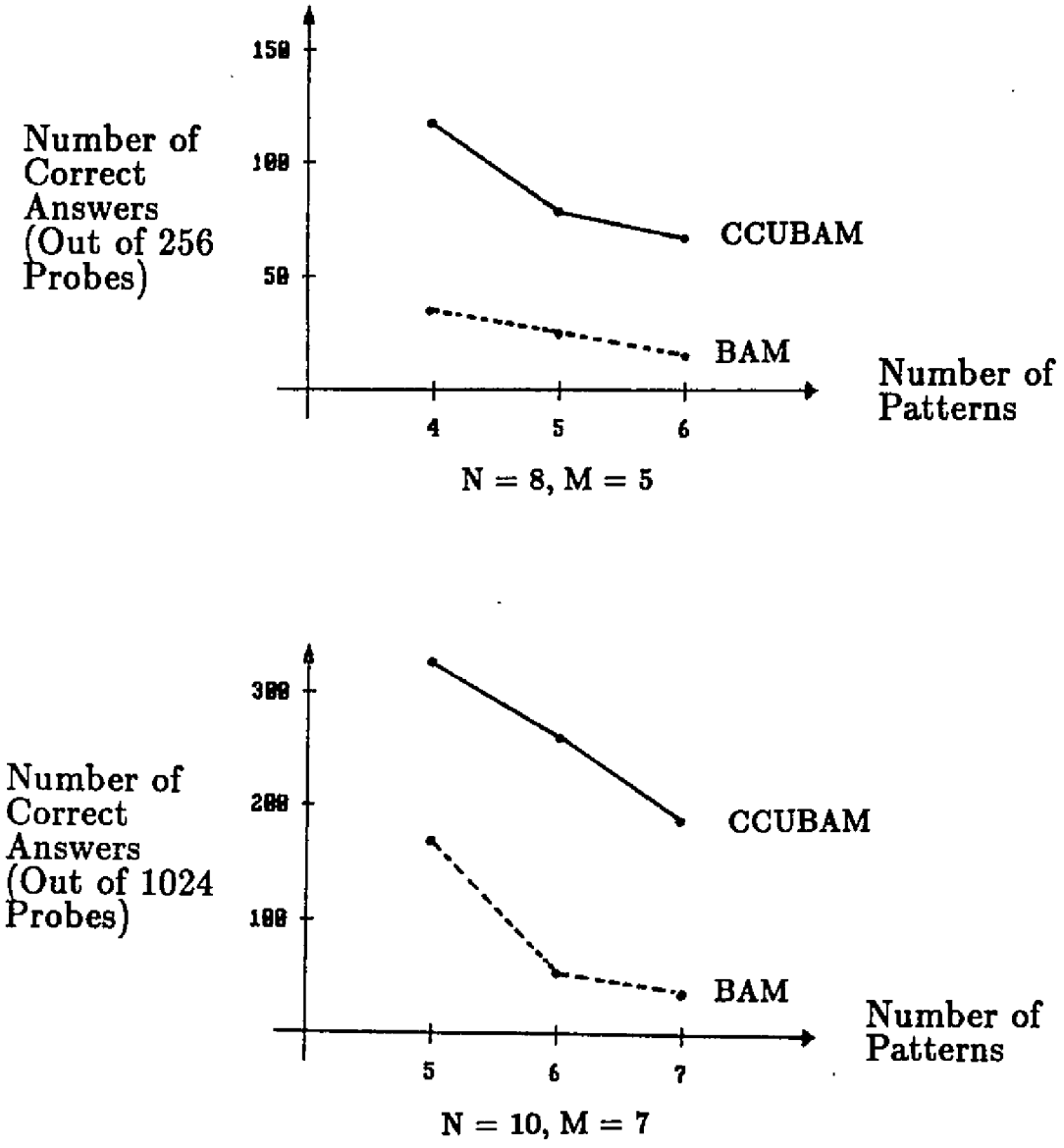


Figure 17

Performance Comparison of BAM and CCUBAM

to train multi-layered networks to perform mapping between input and output patterns. First we describe the well-known backpropagation algorithm [Rume86].

6.3.1. Backpropagation Algorithm

This algorithm is a generalization of the Delta rule described in chapter 2. However, the implementation of this generalized Delta rule requires the use of a differentiable (semi-linear) activation function.

1) Activation Function

In the McCulloch-Pitts model [McCu43], each neuron performs a thresholding function. It hard-limits the output of a neuron to +1 or -1 by comparing the weighted sum, $\sum_i T_{ji} * O_i$, to its threshold, which is zero. But, in the layered network, the use of hard-limiting creates problems, because in the hidden neuron, the information which comes from the previous layer may be lost by this hard-limiting. Thus we need an activation function, F, which is continuous, nondecreasing and differentiable and the output of a neuron, $F(\sum_i T_{ji} * O_i)$, should be real-valued in the range (0, 1). In the following, we define the output of a neuron as O_i, and the activation function as F. This semi-linearity is essential to the function of

a layered network, as will be explained later. Figure 18 represents the activation function of the McCulloch-Pitts neuron. Note that it is discontinuous. The logistic activation function, O_j, is a popular differentiable function that has been used in [Rume86], which is shown in Figure 19. Another function that can be used for this purpose is tanh(x).

$$O_j = \frac{1}{1 + e^{-(\sum_i T_{ji} * O_i + \theta_j)}}$$

where θ_j represents the bias that is similar to the threshold.

In addition to the similarity in shape to Figure 18, it has desirable properties such as continuity and differentiability. Also it produces the real-valued output in the range (0, 1).

2) Weight Matrix

The Hopfield model requires only one connection matrix T, but the layered network needs more than one T. If we restrict ourselves to networks with a single middle layer, we have two sets of T's, one for connections between the input and the middle layer and the other for connections between the middle and output layers.

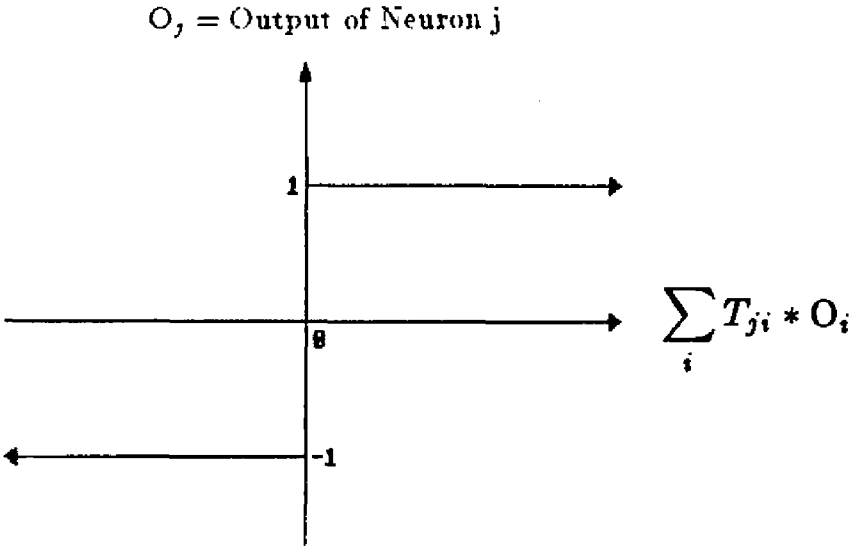


Figure 18

Activation Function of the McCulloch-Pitts Neuron

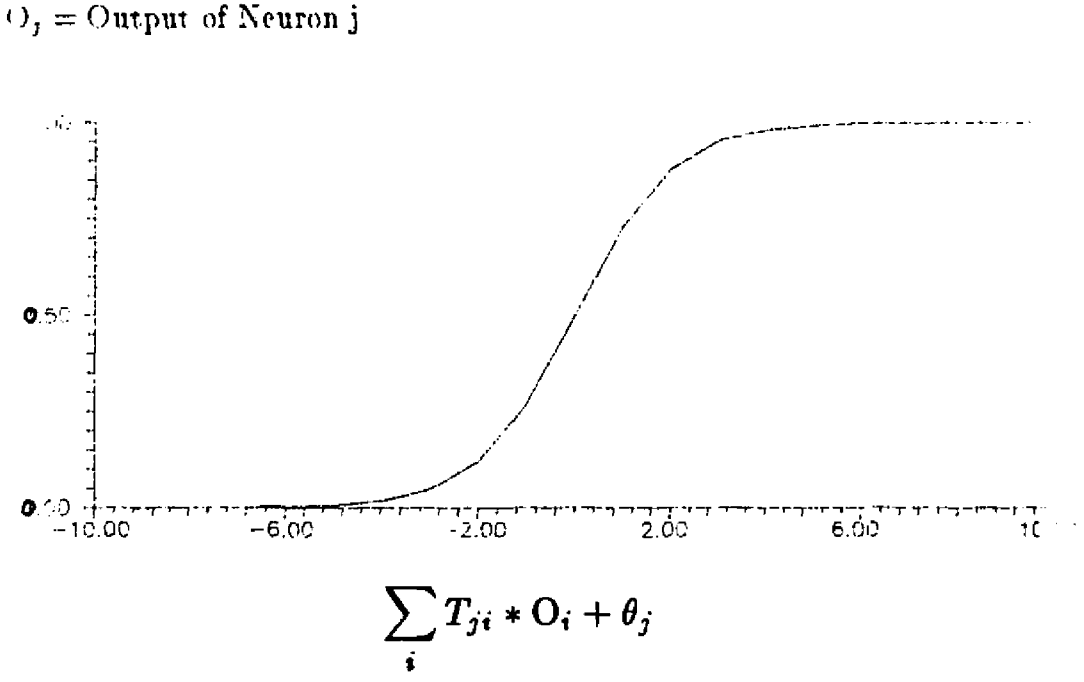


Figure 19

Logistic Activation Function

3) Target Patterns

As in the Delta rule, we should have a target pattern for each input. Since our concern is the application of associative memory, we can use the same input pattern as a target. Here, notice that the logistic activation function reaches 0 and 1 only asymptotically. Therefore, each component of the target pattern can be either 0.1 or 0.9, denoting 0 or 1 respectively. Note that only the output neurons can have a target pattern. Since hidden neurons do not have target values, we need to approximate them. Next we derive the equations which update the weight matrices.

Suppose that Figure 20 describes the surrounding of the neuron j in a layer i. If we define $\sum_i T_{ji} * O_i$ as NET_j, O_j becomes $F(NET_j)$. Then,

$$\frac{\partial NET_j}{\partial T_{ji}} = O_i$$

Also define the squared sum of error, E, as $\frac{1}{2} * \sum_j (t_j - O_j)^2$ where t_j is the target value of neuron j. Another term δ_j is defined as $-\frac{\partial E}{\partial NET_j}$.

By the chain rule,

$$\frac{\partial E}{\partial T_{ji}} = \frac{\partial E}{\partial NET_j} * \frac{\partial NET_j}{\partial T_{ji}}$$
$$= -\delta_j * O_i$$

Also, the amount of change in the weight matrix T can be expressed as

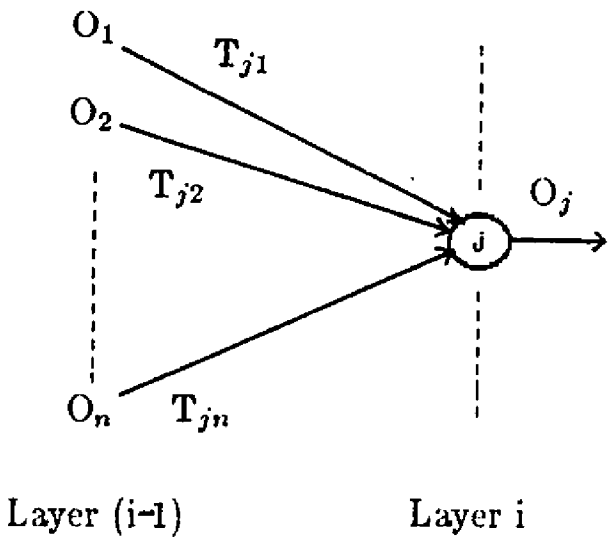


Figure 20

Neurons in Multilayered Feedforward Network

$$T_{ji} = L * (t_j - O_j) * O_i$$
$$= L * \delta_j * O_i$$

where L is the learning rate.

Then we have

$$\delta_j = \frac{-\partial E}{\partial NET_j}$$
$$= \frac{-\partial E}{\partial O_j} * \frac{\partial O_j}{\partial NET_j}$$
$$= \frac{-\partial E}{\partial O_j} * F_j'(NET_j)$$

Note that target patterns are only available in the output layer. Therefore, we should have two different formulas for δ_j , one for neurons in the output layer and the other for layers with hidden neurons.

1) Output Neuron

Since we have the target pattern,

$$\frac{-\partial E}{\partial O_j} = \frac{-\partial}{\partial O_j}[\frac{1}{2} * \sum_j (t_j - O_j)^2]$$
$$= t_j - O_j$$

Therefore, $\delta_j = (t_j - O_j) * F_j'(NET_j)$.

2) Hidden Neuron

In this case, we have to use the chain rule in the computation of δ_j .

$$\delta_j = \sum_k \frac{\partial E}{\partial NET_k} * \frac{\partial NET_k}{\partial O_j}$$

$$= \sum_k \frac{\partial E}{\partial NET_k} * \frac{\partial}{\partial O_j}[\sum_i T_{ki} * O_i]$$

$$= \sum_k \frac{\partial E}{\partial NET_k} * T_{kj}$$

$$= -\sum_k \delta_k * T_{kj}$$

Therefore,

$$\delta_j = \frac{-\partial E}{\partial O_j} * F_j'(NET_j)$$

$$= F_j'(NET_j) \sum_k \delta_k * T_{kj}$$

If we take the logistic activation function as F for all neurons, except neurons in the input layer that pass the input pattern to the next layer, we can compute $F_j'(NET_j)$.

$$F_j'(NET_j) = \frac{\partial O_j}{\partial NET_j}$$

$$= \frac{d}{dNET_j}\left[\frac{1}{1+e^{-NET_j}}\right]$$

$$= \frac{e^{-NET_j}}{(1+e^{-NET_j})^2}$$

$$= \frac{1}{1+e^{-NET_j}} * \frac{e^{-NET_j}}{1+e^{-NET_j}}$$

$$= O_j * (1 - O_j)$$

Therefore,

$$\delta_j = (t_j - O_j) * O_j * (1 - O_j) \qquad \text{--- (1)}$$

for output neurons.

$$\delta_j = O_j * (1 - O_j) * \sum_k \delta_k * T_{kj} \qquad \text{--- (2)}$$

for hidden neurons.

These two equations provide the recursive procedure for the computation of δ_j . Note that the equation (2) uses δ_k's computed in the output layer using equation (1). Basically, the backpropagation algorithm is the same as the Delta rule, except for the computation of the error term in the hidden layer.

6.3.2. Layered Network as an Associative Memory

Our main interest in the multi-layered network is to examine the application of an associative memory. We set up the model so that input patterns are the same as target patterns. The stopping criterion is when the total sum of squared error term, $\sum\sum_j (t_j - O_j)^2$, is less than a given value. We used 0.01 and 0.04 as the critical value for the stopping condition in our experiment. The following is the result of our experiment.

For our examples, we set up a network with three layers, input, hidden and output. All layers have the same number of neurons and we used the logistic activation function for all hidden and output neurons.

1) We were successful in storing up to 2^N patterns, where N is the number of neurons in the input pattern. The comparison of capacity to the previous Hebbian, Delta and continuous unlearning rule cannot be done in a

straightforward manner, because the Hopfield model has a different topology. We find that as the number of patterns increases, the capability of error correction decreases. As for the associative memory, this error correction is an important characteristic. Therefore, the network with its full capacity (i.e. 2^N) is not interesting. In our experiments, we store about N patterns and examine the characteristics of output patterns.

2) Whereas the input pattern has 0 or 1 as a component, the actual output pattern from the output layer produces real-valued components between 0.0 and 1.0. So we have to convert these real-valued components to the corresponding integers, 0 or 1. The question of a cut-off point arises. Generally, input patterns produce strong outputs in the sense that they are within the range of 0.05 from 0 or 1. In our experimentation, we used 0.5 as the cut-off point and determined the outcome.

3) Generally we need more iterations (i.e., backpropagations) as the number of input patterns increases. There are two ways of performing backpropagation. One way is to update the weight matrices whenever the input pattern differs from the target pattern. In this method, we process one pattern at a time. The other is to update the weight matrices only after

computing δ_j 's for all input patterns. In our examples, we used the former method, because it converges faster than the former method.

Example:

Number of input neurons = 5
Number of hidden neurons = 5
Number of output neurons = 5

Number of Input Patterns	Total Number of iterations
5	412
6	454
7	593
9	532
10	675

Rumelhart mentioned the use of the momentum term to increase the learning rate without leading to oscillation [Rume86]. The momentum term utilizes the previous weight change in the next update to improve the learning rate.

4) We find out that this layered approach also suffers from the same problem of spurious patterns as in the Hopfield model. Among them, there are complements to the original input patterns, in addition to non-complement spurious patterns. In the asynchronous Hopfield model, we can force the order of updates using the bicameral approach to improve the convergence. But in this layered

network, we cannot control the inner operation. In other words, it cannot operate asynchronously.

Example:

Number of input neurons = 5
Number of hidden neurons = 5
Number of output neurons = 5
Number of input patterns = 5

X_1 = (0, 0, 0, 0, 0)
X_2^1 = (1, 0, 1, 1, 0)
X_3^2 = (0, 1, 0, 1, 1)
X_4^3 = (1, 1, 1, 0, 0)
X_5^4 = (1, 0, 1, 0, 1)

List of Spurious Patterns

Actual Outputs	Spurious Patterns
(0.81, 0.01, 0.81, 0.02, 0.02)	(1, 0, 1, 0, 0)
(0.19, 0.99, 0.16, 0.96, 0.02)	(0, 1, 0, 1, 0)
(0.31, 0.98, 0.33, 0.05, 0.85)	(0, 1, 0, 0, 1)
(0.05, 0.48, 0.06, 0.96, 0.95)	(0, 0, 0, 1, 1)
(0.93, 0.90, 0.93, 0.83, 0.94)	(1, 1, 1, 1, 1)
-------Complements	
(0.98, 0.18, 0.98, 0.87, 0.94)	(1, 0, 1, 1, 1)
(0.01, 0.98, 0.01, 0.03, 0.05)	(0, 1, 0, 0, 0)
(0.98, 0.97, 0.98, 0.08, 0.88)	(1, 1, 1, 0, 1)
(0.04, 0.13, 0.04, 0.98, 0.04)	(0, 0, 0, 1, 0)
(0.96, 0.96, 0.95, 0.89, 0.02)	(1, 1, 1, 1, 0)
(0.02, 0.09, 0.03, 0.03, 0.94)	(0, 0, 0, 0, 1)
-------Non-Complements	

6.3.3. Indexing of Patterns

The importance of data coding in the neural network was mentioned in [Prad88c, Pott87]. By changing input patterns in such a way that increases the Hamming distances among patterns, we could increase the capacity

of the network. Another way of achieving this is to introduce redundancy. The methods of hidden-bit and orthogonal handles belong to this category [Stin88a, Stin88b]. They are introduced to improve the convergence with the help of a controller. Stinson and Kak presented a new mechanism for memory and recall, wherein a key suffices to retrieve a memory [Stin88b]. It can be used to develop a database management system (DBMS). By appending a key which is comprised of two components: (1) one bit to distinguish a valid memory from its complement, and (2) an orthogonal handle, one can retrieve an original memory when presenting only the key information. During query processing, the locations which represent key bits are not allowed to be updated. But, in this method, the user must be aware of the exact key to retrieve information. In other words, there is no error correction capability. Kak proposed an indexing scheme that can store and retrieve an arbitrary number of related patterns using a bicameral network [Kak89b]. In his method, each set of related patterns is concatenated to form a single binary memory: memory fragments may then be used to recover the entire memory.

As shown in previous sections, a multi-layered feedforward network can be used to map input patterns to target patterns. We propose to employ "indexing" idea using networks with hidden neurons. The basic idea is to

structure target patterns in an orthogonal manner and train the network so that an input pattern converges to the corresponding orthogonal target pattern. It is shown to improve convergence and does not have problems of inherent spurious memories.

Example:

Number of input neurons = 5
Number of hidden neurons = 5
Number of output neurons = 5
Number of input patterns = 5

```
( 0, 0, 0, 0, 0) ----> ( 1, 0, 0, 0, 0)
( 1, 0, 1, 1, 0) ----> ( 0, 1, 0, 0, 0)
( 0, 1, 0, 1, 1) ----> ( 0, 0, 1, 0, 0)
( 1, 1, 1, 0, 0) ----> ( 0, 0, 0, 1, 0)
( 1, 0, 1, 0, 1) ----> ( 0, 0, 0, 0, 1)

 Input Patterns          Target Patterns
```

After learning is completed, the input probe is presented to the network and the output will be one of the orthogonal patterns. In this method, there is one disadvantage. Since there is a single 1 in the output, we can consider the location of 1 in the output as an index to the correct pattern. Therefore, we need a table of patterns to look up. We consider that this requirement is not a serious problem, because the current hardware cost is low enough to afford an additional table in the memory. There are several aspects to consider in this method.

1) Similar to the situation discussed in Section 6.3.2, we have to determine the cut-off point for the output.

2) Instead of having spurious memories, we can have invalid outputs. The valid output should have a single 1 with the remaining bits being all zero's. During our experiments, we found several invalid outputs. Most of them do not have 1's in the outputs.

Let C = the number of probes which lead to the correct pattern, W = the number of probes which lead to the wrong pattern, and Invalid = the number of probes which lead to an invalid output.

Number of Neurons			Number of			Invalid	
Input	Hidden	Output	Patterns	C	W	No 1's	More than one 1
5	5	5	5	27	1	4	0
		6	6	22	4	6	0
		7	7	17	2	8	5
		8	8	18	3	7	4
		9	9	18	3	6	5
6	6	6	6	47	2	12	3
		7	7	41	2	17	4
		8	8	42	7	8	7
		9	9	38	2	23	1

3) For those invalid outputs, we can use a controller to improve performance as in the bicameral network. The

controller picks the largest component out of the invalid output and use it as an index to the pattern. This addition significantly enhances performance.

Number of Neurons			Number of		
Input	Hidden	Output	Patterns	C	W
		5	5	28	4
		6	6	27	5
5	5	7	7	28	4
		8	8	24	8
		9	9	27	5
		6	6	57	7
		7	7	58	6
6	6	8	8	49	15
		9	9	56	8

Chapter Seven

Conclusions

This dissertation has presented new methods to improve the performance of neural networks. A new learning algorithm, CCU, has been proposed that creates a better structure for fixed points and improves performance when compared to the existing learning rules. Two new control methods have been presented to enhance the convergence characteristics. One is the method of iteration, and the other is the bicameral network approach, which uses one part of the network as an asynchronous controller. These methods are applied to two different information models: in one of these, all patterns have the same probability of occurrence and in the other, they do not.

The CCU rule is a technique that tries to remove undesired spurious memory whenever it is encountered during the learning period. This method shrinks the attraction basins around spurious memories and improves the probability of converging to an useful state.

The method of iteration uses the Hamming distance between the probe and the output as a criterion to stop the iteration. This method almost always guarantees the final result within the given distance from the input probe, but this result could be a spurious memory. Two

methods may be used to determine the nature of the output: the Hidden-bit and the Handle. The Hidden-bit detects such spurious outcomes that are complements of original fundamental memories. The Handle identifies spurious memories other than complements.

The bicameral network employs existing clustering techniques, exploits the information about the resulting classes and helps the neural network to converge to correct answers.

7.1. Neural Network System for Speech & Vision Problems

Many researchers have investigated the use of artificial neural network for speech understanding and vision problems. The capability of a neural network to handle incomplete (or noisy) information is especially attractive for these problems. Kohonen developed a low-level neural network processor for speech recognition [Koho88]. Carpenter and Grossberg implemented a classifier that can be used to recognize characters in the input examplar [Carp86]. For a review, see [Lipp89]. We describe a general framework of a neural network system that can be used in these applications. First, we discuss inherent problems pertaining to both speech understanding and image processing.

Compared to technological advances in other engineering areas, the progress in these areas is relatively slow because of many difficulties. The ultimate objective of AI research is to achieve a reasoning capability that parallels human information processing. For example, in speech understanding, the perception of each character is only a first step towards understanding of a full sentence. The next step is to recognize a word, but this might need feedbacks from higher-levels, since a word can have many different meanings depending on the context. Therefore we need interactions among different levels in a hierarchy.

Other difficulties include noise and that there exist no normative sound patterns or scenes. Also intonations of a spoken sentence may make the meaning different. We shall now speak in the context of image processing, though the discussion applies to other A. I. areas as well. In image processing, the same object can produce different images depending on the distance from the sensor of a camera to the object, possibly leading to false perception. Therefore the desirable approach is to extract characteristics of an image (or object) regardless of size, instead of processing the actual image by itself.

One well known labelling method for 3-D objects is the Waltz algorithm that is used as a method of constraint satisfaction [Walt75]. By identifying the shape of every

edge, it can be used to recognize trihedral objects in which every vertex represents the conjunction of three planes. But the Waltz procedure can only label trihedral objects.

There are several preprocessing operations that need to be performed before presenting an image to labelling process.

1) Digitization: Each image will be divided into a fixed number of pixels. Depending on the grey level, each pixel has a value representing the darkness.

2) Smoothing: This eliminates unnecessary variations or noise in the image.

3) Segmentation: Group a set of pixels into a cohesive, meaningful blocks.

It may be assumed that this preprocessing is done before presentation to the system.

The structure of a general system is described in Figure 21. We assume a hybrid approach in which the conventional computer and the neural network function together. As shown in the figure, preliminary processing and coding can be performed using a computer. Preliminary processing performs feature extraction such as labelling using the Waltz procedure. Furthermore, when the raw image is presented to the system, smoothing and segmentation can be performed. The output from the preliminary processing is the set of labels.

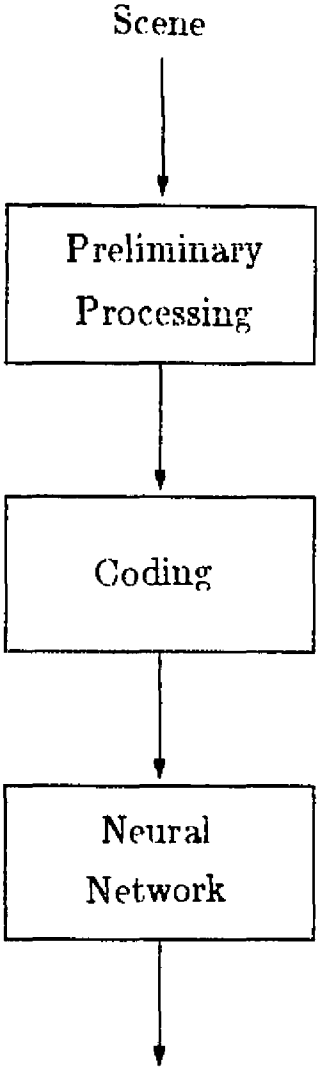


Figure 21

Structure of the Pattern Recognition System

The next step encodes the output from the previous step for the neural network. To encode this information, we have to define the following: 1) the order of vertices in an image, 2) representation of each labelling in an information vector that is processed in the next stage.

This dissertation has only discussed a binary-valued neuron. With a binary neuron, we need 5 neurons to represent all eighteen labellings. Prados and Kak investigated a multi-valued neural network [Prad88c]. This can reduce the number of neurons in an information vector. This coding stage is crucial to the correct function of the system.

In the last stage, the neural network finds the vector that represents the output information. Note that, since we employed preliminary processing, different images of the same object can be handled by the proposed system.

7.2. Directions for Future Research

The asynchronous bicameral network is best viewed as an elementary model that is motivated by a desire to mimic the processing capability of the human brain. As for the controller, there are several aspects of it that can be improved.

1) More refined clustering techniques.

2) Techniques to reduce the size of the configuration without degrading the overall performance. For example, a hashing table can be used to map the original configuration to a shorter one.

Another important question is that of reliability. The performance of a neural network in the presence of different kinds of faults needs to be evaluated.

Owing to the nonlinear nature of neural networks, most studies have been conducted experimentally. It is important to develop an analytical framework in which various different models and algorithms could be compared.

Another area for future study is that of neural networks with "non-classical" synaptic functions. Kak proposed that with nonlinear synaptic functions, one can simulate the chaotic behavior which the human mind appears to go through [Kak88]. It is known that without concentration, a human mind goes through a series of chaotic states. It has been suggested [Kak88] that the process of concentration implies changing a certain parameter which can keep the mind from wandering around meaningless states. So far, synaptic strengths have been represented by scalars. This will have to be generalized and learning algorithms for the new networks will have to be devised.

Human memory is classified into three types: 1) Sensory information storage, 2) Short term memory, 3) Long

term memory. Neural networks that can model all these types may be useful for certain artificial intelligence applications. In particular, the frame problem [Pyly87] requires that information presented to a robot be handled in some hierarchical sense. This would necessitate development of new models of memory and learning.

References

[AbuM85] Y. S. Abu-Mostafa and J. S. Jacques:"Information Capacity of the Hopfield Model," *IEEE Trans. on Information Theory*, Vol. IT-31, No. 4, July 1985.

[Amit87a] D. J. Amit, H. Gutfreund and H. Sompolinsky: "Information storage in neural networks with low levels of activity," *Physical Review A*, Vol. 35, No. 5, pp. 2293-2303, 1987.

[Amit87b] D. J. Amit, H. Gutfreund and H. Sompolinsky: "Statistical Mechanics of Neural Networks near Saturation," *Journal of Physics*, 173, pp. 30-67, 1987.

[Barn88] E. Barnard and D. Casasent:"A Comparison between Criterion Functions for Linear Classifiers, with an Application to Neural Nets," Unpublished paper, 1988.

[Carp86] G. A. Carpenter and S. Grossberg: "Neural Dynamics of Category Learning and Recognition: Attention, Memory Consolidation, and Amnesia," in *Brain Structure, Learning, and Memory*, edited by J. Davis, R. Newburgh, and E. Wegman, AAAS Symposium Series, 1986.

[Cric83] F. Crick and G. Mitchison:"The function of dream sleep," *Nature*, vol. 304, pp. 111-114, July 1983.

[Duda73] R. O. Duda and P. E. Hart:*Pattern Classification and Scene Analysis*, John Wiley & Sons, New York, 1973.

[Gros88] S. Grossberg: *Neural Networks and Natural Intelligence*, The MIT Press, Cambridge, pp. 1-54.

[Hamm82] R. W. Hamming: *Coding and Information Theory*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1982

[Hart75] J. A. Hartigan:*Clustering Algorithms*, John Wiley & Sons, New York, 1975.

[Hebb49] D. O. Hebb: *The Organization of Behavior*, John Wiley & Sons, New York, 1949.

[Hopf82] J. J. Hopfield: "Neural networks and physical systems with emergent collective computational abilities," *Proc. Nat. Acad. Sci. USA*, vol. 79, pp. 2554-2558, 1982.

[Hopf83] J. J. Hopfield et. al.: "Unlearning has a stabilizing effect in collective memories," *Nature*, vol. 304, pp. 158-159, July 1983.

[Hube62] D. N. Hubel and T. N. Wiesel: "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *Journal of Physiology*, Vol. 160, pp. 106-154, 1962.

[Hube74] D. N. Hubel and T. N. Wiesel: "Sequence regularity and geometry of orientation columns in the monkey striate cortex," *Journal of Comparative Neurology*, Vol. 158, pp. 267-294, 1974.

[Jack85] P. C. Jackson: *Introduction To Artificial Intelligence*, Dover Publications, New York, 1985.

[Kak88] S. C. Kak: "Chaotic States of Mind-Neural Networks with Nonclassical Synaptic Function," LSU Technical Report, ECE, May 1988.

[Kak89a] S. C. Kak and M. C. Stinson: "Bicameral Neural Network where Information can be indexed," *Electronics Letters*, Vol. 25, No. 3, pp. 203-205, 1989.

[Kak89b] S. C. Kak: "Storing Pattern Sets in a Bicameral Neural Network," LSU Technical Report #89-003, March, 1989.

[Kant87] I. Kanter and H. Sompolinsky:"Associative recall of memory without errors," *Physical Review A*, Vol. 35, No. 1, pp. 380-392, 1987.

[Koho84] T. Kohonen: *Self-Organization and Associative Memory*, Berlin: Springer-Verlag, 1984.

[Koho88] T. Kohonen: "The 'neural' phonetic typewriter," *Computer*, Vol. 21, 1988, pp. 11-22.

[Kolm57] A. N. Kolmogorov: "On the Representation of Continuous Functions of Many Variables by Superposition of Continuous Functions of One Variable and Addition," *Dokl. Akad. Nauk USSR*, 114, pp. 953-956, 1957.

[Kosk87] B. Kosko: "Constructing an Associative Memory," *BYTE*, pp. 137-144, September, 1987.

[Kosk88] B. Kosko: "Bidirectional Associative Memories," *IEEE Trans. on Systems, Man and Cybernetics*, Vol. 18, No. 1, pp. 49-60, February, 1988.

[Lipp87] R. P. Lippmann: "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, pp. 4-22, April 1987.

[Lipp89] R. P. Lippmann: "Review of Neural Networks for Speech Recognition," *Neural Computation*, Vol. 9, pp. 1-38, 1989.

[McCu43] W. S. McCulloch and W. Pitts:"A Logical Calculus of the Ideas Imminent in Nervous Activity," *Bulletin of Mathematical Biophysics*, Vol. 5, pp. 115-133, 1943.

[McEl87] R. J. McEliece, E. C. Posner, E. R. Rodemich and S. S. Venkatesh: "The Capacity of the Hopfield Associative Memory," *IEEE Trans. on Information Theory*, Vol. IT-33, No. 4, July 1987.

[Mins88] M. L. Minsky and S. A. Papert: *Perceptrons*, The MIT Press, Cambridge, MA, 1988.

[Pott87] T. W. Potter: "Storing and retrieving Data in a Parallel Distributed Memory System," Ph.D. Dissertation, SUNY at Binghamton, 1987.

[Prad88a] D. L. Prados:"The Capacity of a Neural Network," *Electronics Letters*, Vol. 24, pp. 454-455, 1988.

[Prad88b] D. L. Prados and S. C. Kak: "Shift invariant associative memory," in *VLSI for Artificial Intelligence*, edited by J. D. Delgado-Frias and W. R. Moore, Boston: Kluwer, pp. 189-197, 1989.

[Prad88c] D. L. Prados and S. C. Kak: "Non-Binary Neural Networks," *Proceedings of the ComCon 88*, Baton Rouge, pp. 747-754, October, 1988.

[Prad89] D. L. Prados and S. C. Kak: "Neural Network Capacity Using the Delta-rule," *Electronics Letters*, Vol. 25, No. 3, pp. 197-199, 1989.

[Pyly87] Z. W. Pylyshyn; editor: *The Robot's Dilemma*, Ablex Publishing, Norwood, NJ, 1987.

[Rume86] D. E. Rumelhart and J. L. McClelland: *Parallel Distributed Processing*, The MIT Press, Cambridge, 1986.

[Spri81] S. P. Springer and G. Deutsch: *Left Brain, Right Brain*, W. H. Freeman & Co., San Francisco, 1981.

[Stin88a] M. C. Stinson and S. C. Kak: "Asynchronous controller to improve the convergence of neural nets," *Proceedings of the 26th Annual ACM Conference*, Mobile, pp. 410-413, April 1988.

[Stin88b] M. C. Stinson and S. C. Kak: "On Bicameral Neurocomputing," LSU Technical Report #88-90, 1988.

[Walt75] D. L. Waltz: "Understanding Line Drawings of Scenes with Shadows," in *The Psychology of Computer Vision*, edited by P. Winston, McGraw-Hill, New York, 1975.

[Widr60] G. Widrow and M. E. Hoff: "Adaptive Switching Circuits," *1960 IRE WESCON Conv. Record*, Part 4, pp. 96-104, August 1960.

[Youn88a] C. H. Youn and S. C. Kak:"New Learning & Control Algorithms for Neural Networks," *Proceedings of the ComCon 88*, Baton Rouge, pp. 711-722, October 1988. Also to appear in *Advances in Computing and Control*, edited by W. A. Porter, S. C. Kak, and J. L. Aravena, Springer-Verlag, Heidelberg, 1989.

[Youn88b] C. H. Youn and S. C. Kak:"New Learning & Control Algorithms for Neural Networks," LSU Technical Report, #88-052, October, 1988.

[Youn89] C. H. Youn and S. C. Kak: "Continuous Unlearning In Neural Networks," *Electronics Letters*, Vol. 25, No. 3, pp. 202-203, 1989.

VITA

Chung Hwa Youn is the son of Han Chae Youn and Ae Ran P., Youn. He was born in Seoul, Korea, on Februrary 2, 1954. He entered Kyunggi High School and graduated in February 1972. He obtained a B.S. Degree in Mathematics with major in Linear Algebra from the Seoul National University in August 1979. During his undergraduate years, he served in the Republic of Korea Air Force for three years from July 1973 to May 1976. He began his graduate study in the Department of Computer Science at the University of Texas at Austin in August 1980, and graduated with a B.A. Degree in December 1984. He began his Ph.D. program at the Department of Computer Science in Louisiana State University in August 1985.

He is married to Young Hee Youn and they have a son, Suk Min, and a daughter, Hee Yun. Currently, he is a candidate for the Doctor of Philosophy degree at the Department of Computer Scienece in Louisiana State University, Baton Rouge, Louisiana.

DOCTORAL EXAMINATION AND DISSERTATION REPORT

Candidate: Chung Hwa Youn

Major Field: Computer Science

Title of Dissertation: New Learning and Control Algorithms for Neural Networks

Approved:

Subhash Kak
Major Professor and Chairman

F. Glen Hamby
Dean of the Graduate School

EXAMINING COMMITTEE:

Steven F. Lan

David K. Fy

Lakshmanamurthy

H. L. Carmen

Yanbo Chen

Date of Examination:

June 8, 1989